KAI RICHTER

# Compositional Scheduling Analysis Using Standard Event Models

The SymTA/S Approach

**Dissertation, 2005**

Institute of Computer and Communication Network Engineering
Department of Electrical Engineering and Information Technology
Technical University Carolo-Wilhelmina of Braunschweig
Braunschweig, Germany

**KAI RICHTER**

# Compositional Scheduling Analysis Using Standard Event Models

The SymTA/S Approach

**Dissertation, 2005**

Institute of Computer and Communication Network Engineering
Department of Electrical Engineering and Information Technology
Technical University Carolo-Wilhelmina of Braunschweig
Braunschweig, Germany

# Compositional Scheduling Analysis Using Standard Event Models

**(Kompositionelle Schedulinganalyse mit Standard-Ereignismodellen)**

Von der Gemeinsamen Fakultät für Maschinenbau und Elektrotechnik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation


| | |
|---|---|
| eingereicht am: | 25. Oktober 2004 |
| mündliche Prüfung am: | 22. Dezember 2004 |



| | |
|---|---|
| von: | Dipl.-Ing. Kai Robert Richter |
| aus: | Northeim, Bundesrepublik Deutschland |



| | |
|---|---|
| Referent: | Prof. Dr.-Ing. Rolf Ernst |
| Referent: | Prof. Dr. Petru Eles |
| Vorsitzender: | Prof. Dr. Jörn-Uwe Varchmin |


**2005**

# COMPOSITIONAL SCHEDULING ANALYSIS USING STANDARD EVENT MODELS

## The SymTA/S Approach

KAI RICHTER
Institute of Computer and Communication Network Engineering
Department of Electrical Engineering and Information Technology
Technical University of Braunschweig
Braunschweig, Germany

# Abstract

The ever increasing advances in silicon and communication technology allow the integration of increasing functionality in digital embedded systems, ranging from mobile phones through multimedia home platforms to automotive control networks. Embedded system designers have to cope with this increasing system complexity whilst still building competitive systems to survive the market pressure, thus requiring increasing productivity. Component and sub-system specialization and re-use provide this productivity, but system integration is becoming a new bottleneck because the resulting heterogeneous system must be analyzed and verified. This thesis focuses on the performance and timing verification aspect.

Many embedded systems are real-time systems and must meet a variety of timing requirements, such as deadlines and limited load or bandwidth, under all possible corner-case operating conditions. But verification is difficult as timing properties depend heavily on interactions between tasks in the system and on the scheduling of individual tasks and communications. Unfortunately, the current practice of specialization and re-use results in increasingly heterogeneous systems, which specifically complicates the scheduling analysis problem. Even today's best practice of timed simulation is increasingly unreliable, mainly because the corner cases are extremely difficult to find and debug, and it is even more difficult to find simulation patterns to cover them all.

As an alternative, formal scheduling analysis techniques systematically analyze corner cases and provide guaranteed bounds on certain timing properties. Many approaches have been proposed, but most of them are limited to subproblems, such as a single operating system. They ignore the complex, heterogeneous nature of today's applications and architectures, and they cannot be reasonably combined because they use different underlying analysis models. Few approaches target more complex and heterogeneous systems and have their application in specific areas, such as network processor design. These approaches, however, are often too unwieldy and complex to be accepted or

understood by the embedded system industry. We summarize that none of the existing techniques is sufficiently general to comprehensively cover arbitrary heterogeneous systems.

The main objective of this thesis is to develop a scheduling analysis procedure that a) can cope with the increasing complexity and heterogeneity of embedded systems, b) provides the modularity and flexibility that the established, re-use driven system integration style requires, and c) facilitates system integration using a comprehensible analytical model.

In this thesis, a novel, structured analysis model is presented, that elegantly captures the system-level component interactions using intuitive event models. These event models represent the interfaces between different components, and the clear interface structure allows their efficient adaptation to different analytical models. Several previously incompatible component and sub-system analysis techniques can now –for the first time– be heterogeneously combined into a system-level analysis. This allows the modular integration of heterogeneous system parts, whilst providing designers with the flexibility to use their preferred local techniques without compromising global scheduling analysis.

Based on this structured analysis model, we define a sound analysis procedure that consists of few comprehensible steps. The procedure is general enough that it can also detect and solve subtle design pitfalls, such as scheduling anomalies or cyclic dependencies, which are virtually impossible to find using simulation. The sound view on the component interactions substantially improves designers' understanding of the key integration problems, and allows them to influence and control these interactions to optimize the system globally.

The application of the approach is demonstrated in detail using a variety of expressive examples and experiments. As the new analysis procedure can be efficiently applied in practice, it provides a serious and promising complement to simulation. It allows comprehensive system integration and optimization, and it provides much more reliable performance analysis results, at the same time requiring far less computation time.

# Kurzfassung

Als Folge des technologischen Fortschritts bei Halbleitern und Kommunikationsmedien werden heute zunehmend mehr Funktionen in digitalen, eingebetteten Systemen integriert. Beispiele reichen von relativ kleinen Mobiltelefonen über Multi-Media Heimgeräte bis hin zu vernetzter Automobilelektronik. Um konkurrenzfähige Produkte zu entwickeln muss die steigende Systemkomplexität bewältigt werden. Die dazu notwendige Entwurfsproduktivität wird vermehrt durch die Wiederverwendung und Integration von spezialisierten Systemkomponenten (Hardware und Software) erreicht, was allerdings zu neuen Engpässen führt, denn die Integration und das resultierende heterogene Gesamtsystem muss analysiert und verifiziert werden. Die vorliegende Arbeit befasst sich mit Kernfragen der Performanz- und Echtzeitanalyse solcher Systeme.

Eingebettete Systeme sind zumeist Echtzeitsysteme und müssen eine Vielzahl von Zeit- und Performanzanforderungen erfüllen, z. B. maximale Reaktionszeiten oder vorgegebene Kommunikationsbandbreiten, und zwar unter Beachtung aller Randfälle (worst-case). Die Echtzeiteigenschaften hängen stark vom Zusammenspiel der Einzelkomponenten sowie deren Scheduling durch Betriebssysteme und Kommunikationsprotokolle ab. Unglücklicherweise führt gerade die Wiederverwendung von spezialisierten Komponenten zu einer Heterogenität, die die Schedulinganalyse zusätzlich erschwert. Als Folge sind die nach dem Stand der Technik eingesetzten Simulationsverfahren zusehends unzuverlässig, da die kritischen Randfälle in der Praxis kaum mehr vollständig bestimmt werden können.

Als Alternative zur Simulation zeichnen sich formale Methoden gerade bei der Betrachtung von Randfällen durch ein systematisches und zuverlässiges Vorgehen aus. Der Großteil existierender Ansätze ist jedoch auf spezielle Teilprobleme beschränkt, z. B. die isolierte Analyse eines Betriebssystems. Aufgrund der unterschiedlichen zugrundeliegenden Analysemodelle sind diese Ansätze inkompatibel zueinander und eignen sich nicht zur Analyse heteroge-

ner Systeme. Einige wenige Ansätze erfassen zwar komplexere Systeme aus speziellen Anwendungsgebieten, z. B. Netzwerkprozessoren, sind jedoch für den Allgemeinfall zu unhandlich und finden daher nur eine geringe Akzeptanz. Zusammenfassend stellen wir fest, dass es heute keine hinreichend allgemeingültige Technik zur umfassenden Schedulinganalyse von heterogenen Systemen gibt.

Das Hauptziel dieser Arbeit ist die Entwicklung eines Verfahrens zur Schedulinganalyse, das a) die steigende Komplexität und Heterogenität angemessen erfasst, b) über die Modularität und Flexibilität verfügt, die mit Wiederverwendung und Integration erforderlich ist, und c) die Integration durch ein nachvollziehbares Analysemodell unterstützt.

Diese Arbeit stellt ein neues, strukturiertes Analysemodell vor, das die komplexen Abhängigkeiten auf der Systemebene mit Hilfe von intuitiven Ereignismodellen erfasst. Diese Ereignismodelle bilden die Schnittstellen zwischen unterschiedlichen Komponenten, und die klare Strukturierung ermöglicht die effiziente Anpassung dieser Schnittstellen an verschiedenartige Analysemodelle. Dadurch können nun erstmals mehrere unterschiedliche, vormals inkompatible Teilsystemanalysen wiederverwendet und unter Beibehaltung der Systemstruktur heterogen zu einer Gesamtsystemanalyse zusammengesetzt werden (Komposition). Dies erlaubt die modulare Integration von unterschiedlichen Systemteilen und gibt Entwicklern gleichzeitig die nötige Flexibilität, ihre bevorzugten lokalen Entwurfsmethoden zu benutzen, ohne auf die globale Schedulinganalyse verzichten zu müssen.

Das Analysemodell bildet die Grundlage für ein allgemein anwendbares Analyseverfahren, das aus wenigen intuitiven Schritten besteht. Das Verfahren ermöglicht Entwicklern, auch diejenigen subtilen Performanzprobleme zu erkennen und aufzulösen, die beispielsweise durch Schedulinganomalien oder komplexe zyklische Abhängigkeiten hervorgerufen werden und in der Simulation praktisch unerkannt bleiben. Durch die Darstellung der Komponenteninteraktionen mittels verständlicher Ereignismodelle können die Kernprobleme der Integration nachvollzogen werden. Weiterhin kann gezielt auf die Komponenteninteraktion Einfluss genommen und das System global optimiert werden.

Das Verfahren und die Anwendung wird an einer Reihe aussagekräftiger Beispiele und Experimente im Detail erläutert. Es ist in der Praxis sehr effizient einsetzbar, womit sich eine ernstzunehmende und vielversprechende Ergänzung zur heute etablierten Performanz-Simulation eröffnet. Im Vergleich ermöglicht das neue Verfahren eine umfassende Systemintegration und -optimierung und liefert zuverlässige Analyseergebnisse in deutlich kürzerer Zeit.

# Acknowledgments

This thesis summarizes the key scientific results of my research at the Institute of Computer and Communication Network Engineering (IDA) at the Technical University of Braunschweig, Germany.

First of all, I would like to express my sincere gratitude to my advisor Professor Rolf Ernst for introducing me to embedded real-time systems research, and for supporting me finding my way through. His visionary and challenging ideas –in combination with his patience, insight, and guidance– made this thesis possible and SymTA/S a successful project. I also want to thank Professor Petru Eles for his constructive comments related to my research and my dissertation, and for agreeing to co-examine this work, and Professor Jörn-Uwe Varchmin for chairing the examination committee.

I have had the pleasure to work together with a group of very talented, fun, and challenging people. Many thanks to Dirk Ziegenbein and Marek Jersak, my wonderful colleagues and friends in the SPI project. Marek again for our fruitful ongoing collaboration in the SymTA/S project, and for sharing the boldness to turn it into an entrepreneurial partnership. I also want to thank my junior colleagues Jörn-Christian Braam, Arne Hamann, Rafik Henia, Ruben Jubeh, Razvan Racu, Simon Schliecker, and Jan Staschulat who enriched the SymTA/S project both scientifically and personally. Thanks also to Hans Grönniger and to all other contributors in the SPI and SymTA/S projects.

Further thanks go to the research groups of Professor Lothar Thiele in Zürich and Professor Jürgen Teich in Paderborn/Erlangen, especially to Simon Künzli and Christian Haubelt, and to others for cooperations in many projects.

I want to thank the institute staff for providing and maintaining a professional but still personal work environment, the labs for repeatedly helping me repairing my bike and electronic gadgets, Prof. Ernst again for funding the institute's fully-automatic high-end espresso machine, and Gwylim Jones for proof-reading the manuscript.

I want to thank my parents for recognizing and intensifying my scientific-technical interests early, and for encouraging me to study engineering. Most importantly, I want to thank my beloved and charming Alexandra for her support and understanding for the effort it took me writing up this thesis, for her patience when I was coming home late or working in the week-ends, and for the love and the joy we share in our personal life.

Finally, I like to thank all people who expected to be honored explicitly here. This line is dedicated to you!

# Contents

# List of Figures

# List of Tables

Chapter 1

# INTRODUCTION

There is increasing competition in major digital embedded system markets. Examples range from relatively small system-on-chip (SoC) such as network and telecom processors, through more complex consumer electronics such as multi-media home platforms and mobile appliances, to physically distributed systems in the automotive and avionics domain.

Today, major design goals include high quality and reliability at a competitive cost. Depending on the application, other goals such as low power, size, and weight might also be important. In addition, ever shorter market windows with decreasing product lifetime cycles must be met. As silicon technology advances, more and more functions can be implemented on a single chip. These time-to-market and cost pressures require remarkable *productivity improvements* (design gap).

It is obvious that design from scratch does not provide the amount of productivity required to build competitive systems in time. Industry responds by defining and re-using legacy and IP (intellectual property) components and sub-systems to implement the application functions. These pre-designed and/or supplied components represent the atomic building blocks of today's architectures.

In the area of SoC design, industry offers an increasing number of configurable processor cores, hardware accelerators, peripherals as well as communication infrastructure, protocols, and entire network-on-chip solutions. There is often more than one bus or network on a single chip, and chip design gradually moves from core centric SoC to communication centric MpSoC design.

The same trend can be observed in the area of physically distributed systems, such as automotive, where heterogeneous component integration has already been in practice for several years. Car manufacturers have turned into system houses that integrate hardware and software IP components, including

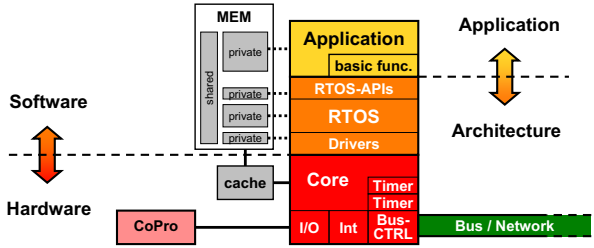



*Figure 1.1.* Complex HW/SW Building Block

operating systems (with OSEK/VDX [80] as a de facto OS standard with many flavors) and so called *basic functions*, or whole electronic control units (ECUs) from different suppliers. The integration is based on standardized communication infrastructure, such as controller area network (CAN [85]), local interconnect network (LIN [72]), the time-triggered protocol (TTP [125]), and Flexray [36].

Figure 1.1 shows a typical building block of today's systems. Almost everything is re-used or supplied externally. Such programmable processors and/or configurable co-processors give ECU designers the amount of flexibility to customize these components quickly with respect to a dedicated distinguishing application. The increasing (re-) use of software functions is a key step for increasing productivity. In turn, software dominated systems require operating systems, APIs, and drivers to control the function execution on shared processors, which further complicates the design process.

This component specialization, optimization, and customization, required to design competitive systems, results in *increasingly heterogeneous* distributed systems with heterogeneous cores, communications, memories, and scheduling strategies. Figure 1.2 shows an example, where the already complex subsystem of Figure 1.1 now becomes merely a small part of a larger system. The integration process is becoming the major challenge including HW/SW component and sub-system interfacing, design space exploration, integration verification, and design process integration. *This dissertation focuses on the verification aspect.*

System verification can be further separated into function verification and architecture performance and timing verification. Function verification is concerned with the correct implementation of a specified function. Roughly speaking, it is verifying that the system calculates the correct output for a given system input. Target architecture performance verification checks if the architecture is able to execute (or perform) a given application, thereby meeting

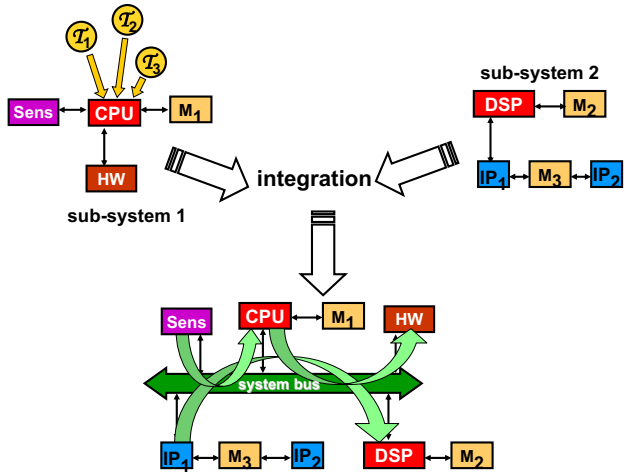



*Figure 1.2.* System Integration using a Shared Bus

a set of non-functional constraints including processing and communication deadlines, memory requirements, etc.. Closely related, function timing analysis provides detailed information about the actual timing of the application, or parts of it. These verification issues are particularly important for all *real-time systems* and require consideration of non-functional performance dependencies. The complexity furthermore increases with system size and architectural hardware/software heterogeneity. *This thesis focuses on performance and timing analysis for heterogeneous architectures.*

## 1.1 Influences on HW/SW System Performance

Complex hardware and software component interactions result in a variety of *performance pitfalls*, including transient overload, buffer under- and overflows, missed deadlines, and architecture dependent dead- or life-locks. The ITRS [104] names system level performance verification as one of the top-three MpSoC design issues. The same problem has been recognized by the "AUTOSAR development partnership" [7], in which large parts of the European automotive industry aim at establishing an open standard for automotive E/E architectures. The leading German electronics magazine [2] says "networking and the increasing software complexity pose key challenges on future automotive system design, and requires re-consideration of integration practice, and cooperations". But why is it so complex?

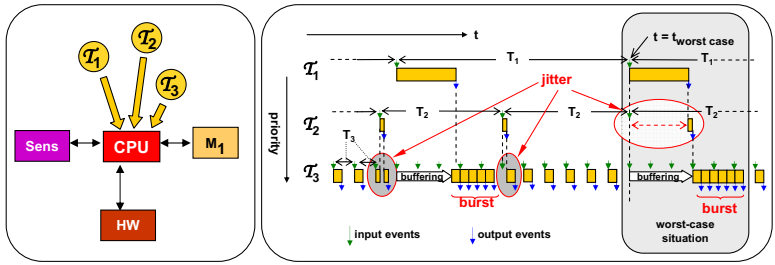



*Figure 1.3.* Complex Execution Sequences

HW and SW integration involves resource sharing based on operating systems and bus/network protocols. Resource sharing requires scheduling that results in a confusing variety of performance dependencies at run-time, which are induced by the architecture and not reflected in the system function.

As an example, Figure 1.3 shows a CPU sub-system executing three independent tasks: $\mathcal{T}_1$ to $\mathcal{T}_3$. Scheduling is preemptive and follows static priorities; a popular OS setup. Although the operating system activates all tasks periodically with periods $T_1$, $T_2$, and $T_3$, respectively, the scheduling diagram in Figure 1.3 shows that the resulting execution sequence is rather complex.

Due to scheduling, the lower priority tasks $\mathcal{T}_2$ and $\mathcal{T}_3$ are preempted or delayed by the higher level task $\mathcal{T}_1$. $\mathcal{T}_2$ can further preempt or delay $\mathcal{T}_3$. Such preemptions and delays disturb the initially periodic task execution. As Figure 1.3 shows, $\mathcal{T}_1$ can noticeably delay the completion time of $\mathcal{T}_2$, resulting in a jitter on $\mathcal{T}_2$'s output. More complexly, $\mathcal{T}_1$ can delay several executions of $\mathcal{T}_3$. After $\mathcal{T}_1$ completes, $\mathcal{T}_3$ –with its input buffers filled– runs in "burst" mode with its execution frequency only limited by the available processor performance. This leads to transient output bursts for $\mathcal{T}_3$, modulated by $\mathcal{T}_1$ execution and possibly further disturbed by $\mathcal{T}_2$ preemptions. A worst-case scenario for $\mathcal{T}_3$ with maximum delay and preemption, maximum buffering, and maximum task response time is highlighted in the figure.

Even this relatively small example with only three tasks, comprehensibly illustrates the tremendous influence of resource sharing on system performance and task timing. Figure 1.3 shows that the influence of scheduling heavily increases with decreasing task priorities. Hence, realistic systems with more (up to 30 and more) tasks and priority levels result in even more complex execution scenarios.

The example above does not even include data-dependent task execution times, which are typical for software systems. Furthermore, operating system overhead is neglected. Both effects have to be considered during analysis and

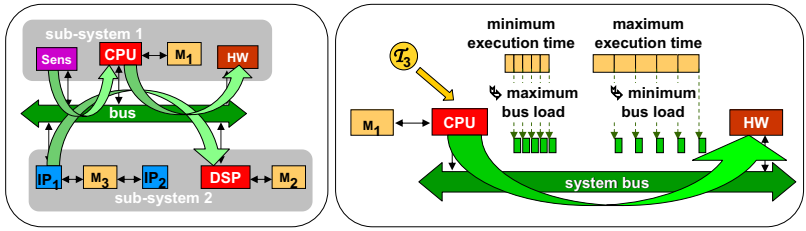



*Figure 1.4.* Scheduling Anomaly in Distributed Systems

further complicate the problem. We conclude that finding corner cases which reveal all influences of scheduling on a single CPU is already challenging.

System integration introduces another type of performance dependencies. Figure 1.4 shows an example. The system of Figure 1.3 is now only a subsystem and is integrated with a second DSP application using a shared bus or network-on-chip (NoC). The curved arrows in Figure 1.4 illustrate performance dependencies between the CPU and DSP sub-systems. These dependencies are not reflected in the system function but result from system integration and network arbitration, required to schedule transmissions over the shared communication bus.

These dependencies can turn component (e. g. CPU) best-case behavior into system worst-case behavior, and vice versa –a so called *scheduling anomaly*. Recall the $\mathcal{T}_3$ burst execution from Figure 1.3 and consider that $\mathcal{T}_3$'s execution time can vary from one execution to the next (data-dependency). There are two critical execution scenarios, called corner cases: The minimum execution time of $\mathcal{T}_3$ corresponds to the maximum transient bus load (number of $\mathcal{T}_3$ outputs and, hence, transmissions over time), slowing down other component's communication. In turn, a maximum execution time results in less frequent data transmissions, and thus represents the minimum bus load during a burst. Larger systems will exhibit even more complex communication patterns.

An additional performance pitfall in distributed system design occurs when *cyclic* performance dependencies are introduced during system integration. These dependencies are subtle and difficult to detect if the integration process does not consider component details. Figure 1.5 highlights a non-functional event stream dependency cycle in the system of Figure 1.4 that is only introduced by communication sharing. Upon receipt of new sensor data, the CPU activates task $\mathcal{T}_1$ which preempts $\mathcal{T}_3$ and thus affects the execution timing of $\mathcal{T}_3$. Figure 1.3 illustrates this preemption. Task $\mathcal{T}_3$'s output, in turn, enters the network on channel $C_2$, where it now interferes with the arriving sensor data on $C_1$. The interference of the two functionally independent channels, $C_1$

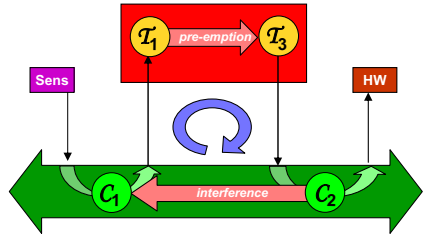



*Figure 1.5.* Non-Functional Performance Dependency Cycles

and $C_2$, closes the dependency cycle, since the sub-system in Figure 1.3 was originally cycle-free.

The complex run-time effects shown in Figures 1.3, 1.4 and 1.5 lead to complex system-level performance corner cases, which, during performance verification, have to be considered and checked against given performance constraints such as buffer constraints, end-to-end deadlines (e.g. from Sensor to HW in Figure 1.4), and many others. Essentially, if all corner cases satisfy the performance constraints, then the system is guaranteed to satisfy the constraints under all possible operation conditions. Conversely, each corner case that is not considered during verification reduces the reliability of the system with respect to performance and timing. However, many corner cases are extremely difficult to find. Just consider the $\mathcal{T}_3$ communication scheduling anomaly in Figure 1.4, and recall that the $\mathcal{T}_3$ bursts further depend on the execution behavior (execution time, activation period) of $\mathcal{T}_1$ (see Figure 1.3).

## 1.2 Performance Simulation and Coverage

Today, test and simulation are the preferred means of function and performance verification. While test can only be applied to the final design or a prototype, simulation is supported at all abstraction levels. High-level languages are often used for first proofs of concept. They allow an *executable specification* of the intended functional behavior, which can be simulated. The interface standardization efforts, such as VSIA [131] and Accellera [1] in the SoC domain, and the mentioned bus protocols and *hardware dependent software* layers that the OSEK/VDX standard [80] defines for automotive systems, supports integration of blocks and components (function and architecture). They target an easy-to-use “cut&paste” design style, since the individual components can be easily pulled from a library and connected to each other. This drastically improves design productivity. Standardized operating systems and APIs have the same goal.

The executable architecture model is then also simulated. Simulation is supported at all levels, ranging from RTL or gate-level to cycle-accurate co-simulation of the entire hardware/software system using tools such as Mentor Seamless CVE [77], or Axys MaxSim [8]. Recently VaST, who started in the SoC and consumer electronics market, extended its CoMET [129] design and simulation environment to also cope with the specific problems of automotive system design. Simulation has some obvious advantages. Using the same simulation environment, the same verification patterns, and benchmarks already available from functional verification, designers have elegant means to test an architecture design against a functional specification.

The co-simulation times are extensive because performance simulation requires detailed timed models that are usually more complex than the un-timed –or simplified– models used in function verification. This is a major bottleneck during design, and becomes particularly painful in iterative design space exploration, reducing optimization possibilities, and consequently, system quality. Abstract simulation using, for instance, Cadence VCC [19], is faster and provides temporary relief but comes at the cost of introducing yet another model. Test is the fastest but can only be applied very late in the design process; and more important than the speed, there is a *serious, conceptual limitation to test- and simulation-based performance verification* that becomes critical as complexity increases.

While function verification can also check for correct functional component (integration) interactions, the complex component performance dependencies that integration, and especially resource sharing introduces, are only partially visible in the system function, if at all. As a consequence, function verification patterns will most likely miss some of the critical performance corner cases illustrated in the preceding section. Identifying *all* performance corner cases is extremely difficult, and it is even more difficult to find patterns that reach all of them.

So, where do we get the patterns from? On the one hand, re-using the patterns from function verification is not sufficient because they do not reflect architecture dependencies that scheduling introduces. On the other hand, component verification patterns –if available– are not sufficient, either, because they do not cover the complex component interactions that result from system integration.

In other words, corner case identification and pattern generation, which are the critical steps in performance simulation, are unfortunately not compositional as the design is. This is a key disadvantage of simulation. If the system is sufficiently simple, an integrator might *manually* add new patterns. But this requires detailed knowledge of component implementations which is often not available to the integrator in this high-level "cut&paste" design style. An additional problem is that corner cases appear and disappear as new components

are added or modified. This shows that manual pattern generation for complex systems with multiple cores and buses is virtually impossible and not a practical option.

Two different reactions to this problem can be observed in industry. First, designers apply more extensive test with more patterns. This is likely to cover more of the mentioned performance corner cases. However, due to the lack of a systematic pattern generation procedure, this apparent improvement does not represent a real advantage as it compromises reliability and increases the design risk. Designers do not know if their patterns cover all corner cases, since there is no means to check this coverage. In effect, test- and simulation-based performance verification is increasingly unreliable as systems grow and we ultimately believe that it will run out of steam soon.

Alternatively, designers can deliberately waive system efficiency and enforce static, synchronous systems [127, 126] using static resource sharing strategies. This is specifically popular for communication infrastructure scheduling as the center of all integration efforts. A variety of new TDMA-based protocols such as TTP [125, 64], Flexray [36], TTCAN [25], and Sonics [107] have been offered in recent years. Such protocols decouple component timing and eliminate a large part of the complex dependencies, so that system-level timing is easily predictable. In other words, the synchronous or static approach benefits from the fact that component and system integration does not add new performance corner cases. It is possible to oversee the individual components, coverage can be manually controlled, and simulation and test can provide very reliable results.

But this simplicity comes at huge efficiency price that increases with system size. There are areas where this is acceptable, such as in the military and avionics industries, where the need for system reliability and dependability dominates other optimization goals such as cost and utilization. However, the conservative and less efficient static approach does not scale well to the high-volume low-cost consumer field with multimedia, telecommunications, mobile appliances, etc., where future MpSoC will integrate multiple heterogeneous OS and complex networks protocols. It does also not meet the requirements of the strongly heterogeneous automotive integration process.

## 1.3 Objectives & Outline

The key intention of our research is to develop a more systematic performance verification procedure for heterogeneous distributed real-time systems, essentially needed to increase system efficiency and quality and to reduce design cost and risk. To be applicable in practice, the technique must consider the established (IP) re-use driven “cut&paste” design style, i. e. it must be compositional and modular.

Formal approaches become attractive as simulation and test fall short. There has been much activity in the real-time systems research community over the last 30 years. Formal performance analysis, such as the popular rate-monotonic analysis [73] (RMA), systematically tackles the corner case problems and is already used in practice. Many of the existing contributions are directly applicable to individual local components of the examples shown earlier in this introduction, some focus on single tasks, some on scheduling, some on networks, etc.. However on the one hand, the approaches do hardly scale as systems grow in size and complexity, and none of them is sufficiently general for arbitrary heterogeneous systems. On the other hand, the underlying models are often mutually incompatible and hence lack the required modularity, which prevents them from being used in system-level analysis. In particular the complex component interactions and the heterogeneous resource sharing (scheduling), that result from system integration, are major sources of analysis complexity. The existing work will be described in detail soon. For this introduction we summarize that formal performance and timing analysis of heterogeneous real-time systems is not possible at present.

This thesis presents a new approach which we call SymTA/S (Symbolic Timing Analysis for System) that provides modular and flexible integration support for components and their underlying scheduling analysis models. Key of the approach is the definition of model interfaces between several heterogeneous sub-system techniques [94]. Such interfaces allow the combination of previously incompatible local models and techniques, which fosters a structured system-level analysis model [101] that can be efficiently and elegantly solved [98]. Known techniques, with their specific limitations and restrictions, can be applied locally to the individual components without compromising global analysis.

The resulting modular system view supports the understanding of complex component interactions, and allows these dependencies to be controlled and optimized during component design and system integration. This enables a novel, comprehensive and reliable system integration procedure. The approach has been proven valuable in the area of network centric SoC [97] and in the automotive domain [56, 95], where operating systems and bus protocols have already been de-facto standardized.

The next chapter provides a broad overview of existing formal scheduling analysis approaches. We specifically discuss the applicability to heterogeneous and distributed systems, and the modularity of the proposed techniques. Through an evaluation of the known work, we refine the work objectives and propose our basic idea. Finally, we identify the main challenges and outline our solution.

In Chapters 3 and 4, we look at the component and task input-output models (*event models*) that popular work on scheduling analysis uses. We highlight

key inefficiencies that prevent their direct modular application in system-level analysis, and we provide a novel structured and self-contained set of *standard event models*. These are the system-level interfaces for our *compositional approach*.

Model incompatibilities, which represent a major challenge with respect to modularity and flexibility, are systematically analyzed in Chapter 5. Appropriate *event model interfaces* are developed. In Chapter 6, we extend the interfaces by so called *event adaptation functions*, that use traffic shaping in order to control and optimize the streams that connect the sub-systems.

Event model interfaces and traffic shaping enable a novel iterative analysis procedure which we present in Chapter 7. A set of experiments demonstrates the application of the approach and validates the underlying ideas. Finally, Chapter 8 summarizes the key contributions of this thesis. Current extensions and future work directions are briefly outlined, before we draw our conclusions.

Chapter 2

# SCHEDULING AND PERFORMANCE ANALYSIS

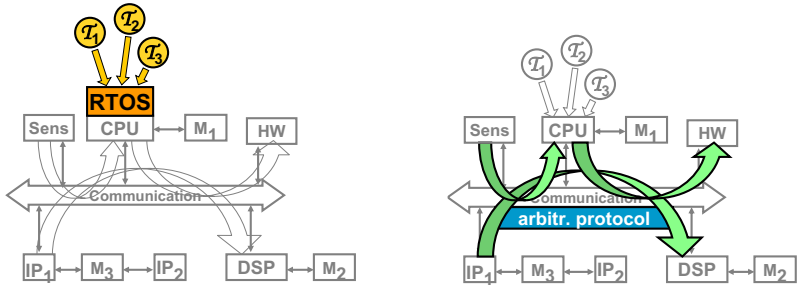
The influence of scheduling on the timing and the performance of embedded applications and HW/SW architectures and platforms has been subject to research for several decades. When dealing with *hard real-time systems*, the goal is to calculate *guaranteed bounds* on timing properties such as execution times and buffering delays as well as processor and network load. A host of formal models and methods has been published. Abstract system models are commonly used to identify corner cases that lead to worst-case and best-case system timing behavior. The sought-after properties are obtained by symbolic simulation or are conservatively calculated.

This chapter summarizes key contributions out of the host of work on timing and specifically scheduling analysis. The survey starts from classical real-time scheduling analysis theory and then targets multiprocessor and system-level extensions, some of them use a quite different view on system-level scheduling. We evaluate the properties of the existing work with respect to the given objectives.

A novel idea for how system-level scheduling analysis can be made more systematic, more comprehensive, and more effective than known approaches is formulated. The key challenges are highlighted to be solved later in this thesis.

## 2.1 Component Scheduling Analysis

Components in this context are individually scheduled sub-systems, usually either a processor or a bus/network. The scope of such component-level analysis is illustrated in Figure 2.1. Since the general concepts are very similar for task and communications scheduling, we do not particularly differentiate between them. The concepts of core time, response time, and load are valid for both, and scheduling analysis can account for operating system and protocol overhead. Rather, we selected few landmark and representative component-




(a) Single-CPU Scheduling Analysis (b) Bus/Network Arbitration Analysis

*Figure 2.1.* Scope of Component-Level Analysis

level scheduling analysis approaches to illustrate their basic common concepts and key differences.

### 2.1.1 Rate-Monotonic Scheduling

In their seminal paper on scheduling analysis, Liu and Layland [73] proposed rate-monotonic scheduling (RMS) as an optimal static priority assignment for independent periodic tasks with deadlines at the end of their periods. Optimal means that no other priority assignment yields better schedulability. RMS assigns static priorities to tasks according to their periods, the smaller the period the higher the priority.

Liu and Layland provided a sufficient schedulability test [73] based on the *processor utilization approach*. They calculate the accumulative load or utilization[1] $U$ of all tasks in the system. The (maximum) load of each task is its maximum execution time $C_i$ divided by its activation period $T_i$. Liu and Layland discovered that such a system with deadlines at the end of the periods is schedulable, if this utilization is below a certain bound that depends on the number of tasks $n$:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{2.1}$$

Exact utilization-based schedulability tests were first proposed by Lehoczky, Sha and others [70, 106]. In general, a successful schedulability test guarantees that a) the system is schedulable, and b) all tasks meet their deadlines which are at the end of the period.

[1] the terms load and utilization are often used interchangeably in literature

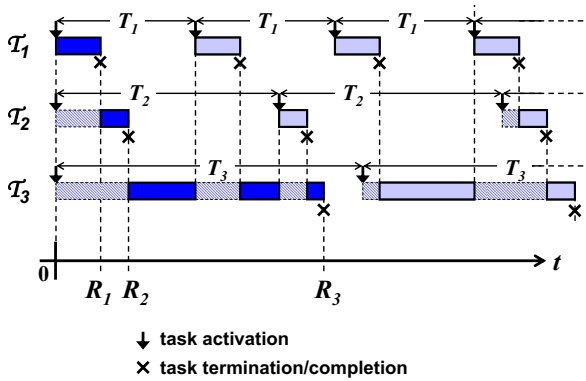



*Figure 2.2.* Scheduling Diagram of Rate-Monotonic Scheduling

Joseph and Pandya [60] developed a first *response time* calculation for RMS. The response time of a task provides more detailed information about the timing of a task than load or utilization models. It represents the externally observable behavior of a task under the influence of scheduling. Joseph and Pandya measured the response time from the task activation to the task completion or termination. Figure 2.2 shows the scheduling diagram of three tasks that are activated periodically. The first response time $R_i$ of each task is shown.

Joseph and Pandya provided a guaranteed *worst-case response time calculation*. They recognized that a task experiences its largest number of preemptions by higher-priority tasks when all tasks are activated simultaneously [60], a situation called the *critical instant* of a task. Figure 2.2 shows the critical instant of all three tasks. Liu and Layland [73] proved that a task that meets its first deadline at its critical instant, will also meet all later deadlines.

The response time of task $\mathcal{T}_i$ is calculated as the sum of the tasks own worst-case execution time $C_i$ plus its worst-case interference $I_i$. The interference term $I_i$ determines how much preemption task $\mathcal{T}_i$ will experience due to higher priority tasks $\mathcal{T}_j$ during it own execution. $\mathrm{hp}(i)$ is the set of higher priority tasks. The deadline $D_i$ is at the end of the period $T_i$.

$$R_i = C_i + \underbrace{\sum_{j \in \mathrm{hp}(i)} C_j \underbrace{\left\lceil \frac{R_i}{T_j} \right\rceil}_{\text{\# of preemptions}}}_{\text{interference term } I_i} \leq D_i = T_i \tag{2.2}$$

Knowledge of the critical instant is very important in schedulability or scheduling analysis. The critical instant is, by definition, a worst-case scenario in the sense that no other scenario will yield a larger response time, or a higher load respectively.

### 2.1.2 Other Static-Priority Schedulers

Since the publication of the first response time approach for RMS [60], a multitude of researchers have extended the applicability, enhanced the accuracy, and improved the efficiency of response time calculations for this and other system set-ups. In the next few paragraphs, we select a few landmark contributions out of this huge field where two groups, Burns and Wellings from the University of York, England, and Lehoczky, Rajkumar (both Carnegie Mellon University) and Sha from the University of Illinois at Urbana-Champaign, can be mentioned as outstanding contributors.

#### 2.1.2.1 Deadline-Monotonic Priority Assignment

For deadlines shorter than periods, Leung and Whitehead [71] proved that the deadline-monotonic scheduling (DMS) approach [69, 5] is the optimal priority assignment, where the task with the shortest deadline is given the highest priority. They enhanced the sufficient load-based schedulability test of Equation 2.1. More interesting in the context of this thesis is that the response time approach [60] of Equation 2.2 can deal with arbitrary priority assignments, so it also applies to DMS.

#### 2.1.2.2 Arbitrary Deadlines

Allowing deadlines larger than periods has conceptual consequences, since tasks can now re-arrive before the previous activation has completed. In other words, tasks can preempt or interfere with themselves, and it is no longer sufficient to check if the first deadline is met. Instead, all activations that lead to self-interference must be checked.

Lehoczky [69] introduced the concept of a *busy period* or *busy window* as a generalization of the concept of a critical instant, that elegantly re-uses the response time approach of Equation 2.2. A busy window captures $q$ consecutive task executions as a single, clustered execution, and a combined response time $w_i(q)$ for such task clusters can be calculated:

$$w_i(q) = \underbrace{q\,C_i}_{\substack{\text{core time of} \\ q \text{ executions}}} + \sum_{j \in \text{hp}(i)} C_j \left\lceil \frac{w_i(q)}{T_j} \right\rceil \tag{2.3}$$

With information about the activation timing, the individual response time of each $q$th execution is calculated following Equation 2.4. For the details, we refer to [69].

$$R_i(q) = \underbrace{w_i(q)}_{\substack{\text{resp. time of} \\ q \text{ executions}}} - \underbrace{(q-1)\,T_i}_{\substack{\text{activation time of} \\ q\text{th execution}}} \tag{2.4}$$

Finally, the overall maximum response $R_i$ time is the maximum of these individual response times $R_i(q)$:

$$R_i = \max_q(R_i(q)) \tag{2.5}$$

#### 2.1.2.3 Extended Task Activation

The model of purely periodic tasks is too static and insufficient to capture reactive embedded system behavior, and other activation principles have been investigated.

**Release Jitter.** Tick-scheduling [4, 121], i. e. a scheduling that periodically tests task activation conditions, can induce a so called *release jitter*. This is the time between the actual arrival time of an activating event, or the time a general activation condition becomes true, and the time this is recognized for scheduling. Tick-schedulers and other polling systems typically induce such delays. Audsley [4] and Tindell [121] investigated the properties of release jitter.

A jitter does not change the *average* period of a task, but the activating events are allowed to deviate with respect to their original period. Hence, in certain circumstances, events can arrive earlier than in case of purely periodic events. This changes the critical instant and effectively increases the number of possible worst-case preemptions. The response-time calculations must consider this [4]:

$$R_i = C_i + \underbrace{\sum_{j \in \mathrm{hp}(i)} C_j \left\lceil \frac{R_i + J_j}{T_j} \right\rceil}_{\text{interference term } I_i} \leq D_i = T_i - J_i \tag{2.6}$$

Compared to Equation 2.2, the jitter $J_j$ adds to the numerator, increases the number of preemptions, and results in a larger response time. Additionally, the deadline must be adjusted accordingly to avoid self-interference, otherwise the windowing techniques (see Section 2.1.2.2) must be applied.

As we will see in Chapter 4, there are other sources of jitter, especially in distributed systems. Scheduling, execution, and communication often delay data for a non-constant time, thereby inducing significant jitter to the output production of tasks, and hence to the activation of successor tasks. Detailed properties of event arrival and task activation models (or event models) will be given in Chapter 3.

**Sporadic Tasks.** Sprunt et. al. [108] differentiated *sporadic tasks* as a special class of *aperiodic tasks* for which the utilization can be bounded. They defined a minimum inter-arrival time $d_i^-$ of task $\mathcal{T}_i$ as a minimum period, corresponding to a maximum allowed frequency [108, 105] . For the purpose of

worst-case response time analysis, sporadic tasks are usually treated as periodic tasks. So, Equation 2.2 as well as Lehoczky's windowing technique also apply to sporadic tasks.

**Sporadic Bursts.** Later, Audsley [4] and Tindell [121] proposed another type of sporadic task activation. The model of *sporadically periodic* events, also called *sporadic bursts*, allows events to be captured that occur temporarily periodically (as a burst) within sporadically bounded distances. Event arrival is bounded to at most $b$ events within an *outer period* of $T^O$. An *inner period* $T^I$ bounds the minimum inter-arrival time between two successive events. Audsley [4] provided the necessary extensions to the response time calculations (see Equation 2.7). Tindell [121] finally applied Lehoczky's windowing technique. The term $B_i$ captures the *blocking time* which is explained in the next section.

$$R_i = C_i + B_i + \sum_{j \in \mathrm{hp}(i)} C_j \left( \min \left( \left\lceil \frac{R_i - T_j^O \left\lfloor \frac{R_i}{T_j^O} \right\rfloor}{T_j^I} \right\rceil, b_j \right) + b_j \left\lfloor \frac{R_i}{T_j^O} \right\rfloor \right) \tag{2.7}$$

#### 2.1.2.4 Shared Resources

Audsley and Burns deeply investigated the impact of shared resources on scheduling and analysis [3]. Shared variables, drivers or any other *critical sections* [113] typically lead to additional delays, not reflected by the priority relation of tasks. A so called *blocking time* $B_i$ was introduced as a correction term that is added to the response time $R_i$. This blocking time captures the maximum delay a task can experience due to critical sections of lower-priority tasks. This was already illustrated in Equation 2.7.

Such shared resources and the use of semaphores and monitors can lead to a situation known as *priority inversion* [105, 3]. Several proposals have been made to reduce the effect of priority inversion, e. g. the priority ceiling protocol [105, 3], the priority inheritance protocol [105, 24], and the kernelized monitor protocol [24].

#### 2.1.2.5 Synchronous, Asynchronous, and Incomplete Task Sets

Most of the above techniques use the idea of a critical instant or busy window in order to determine a single worst-case scheduling scenario. For the rate-monotonic approach as illustrated in Figure 2.2 this leads to *synchronous* task activation. A periodic task set where all tasks are activated simultaneously is referred to as a *synchronous task set* [73].

In an *asynchronous task set*, the tasks need not necessarily be activated simultaneously but with a known or bounded phase delay. Such phase delays are usually called *offsets* [120, 82]. Interestingly such asynchronous task sets are harder to analyze than synchronous task sets. Leung and Whitehead [71] indicated large variations in the algorithmic complexity of seemingly similar scheduling and analysis problems. Baruah et. al. [10] investigated these issues further and found out that much of the schedulability analysis of asynchronous, periodic task sets is NP-hard in the strong sense. However, a set of practically useful approximations exist that perform in polynomial time. Tindell [120] and later Palencia and Harbour et. al. [82, 83] presented a reasonable analytical model that utilizes task offsets to obtain significantly tighter bounds of the response time. They also optimized offsets of tasks according to task precedence relationships due to e. g. communication between tasks.

Finally, there is the class of asynchronous task sets with unknown offsets, called *incomplete task sets* [10], where only the periods are known but offset information is not available. Analysis must assume any choice of offsets. This, interestingly, reduces analysis complexity, since in most situations the worst-case scenario of an incomplete set is identical with that of the corresponding synchronous task set, leading to the known *critical task instant*.

Static-priority based scheduling is one of the most popular fields in real-time systems research. There are countless other publications, each focusing on other aspects, but most of them have the roots presented above. For an extensive overview we refer to three popular books [63, 17, 74]. A compact but incomplete overview paper is available from Fidge [34].

### 2.1.3 Dynamic Priority Assignments

Liu and Layland also provided an analytical model for an optimal dynamic priority assignment policy, called *earliest deadline first* (EDF) [73]. In EDF, the task with the nearest deadline is selected for execution. In contrast to RMS and DMS, the relative priorities between tasks can change dynamically at run-time under EDF scheduling [3], considerably complicating the predictions needed for worst-case analysis. The field of dynamic priorities also knows some key contributors, Stankovic from the University of Virginia in Charlottesville, Buttazzo from Scula Superiore S. Anna in Pisa, Italy, and Jeffay and Baruah from the University of North Carolina at Chappel Hill.

Generally, EDF scheduling analysis suffers from comparable problems to static priority scheduling. Liu and Layland's *utilization approach* for purely periodic task activation [73] is quite simple. A periodic task set is schedulable under EDF, if and only if the following condition is satisfied:

$$\sum_{i=i}^{n} \frac{C_i}{T_i} \leq 1 \tag{2.8}$$

$C_i$ is the core execution time of task $\mathcal{T}_i$ and $T_i$ is its period. First response time approaches were proposed by Spuri [109, 110] and are considerably more complex. In particular the critical instant is no longer easy to determine. Spuri introduced the notation of a *deadline busy period* [110] to ease the response time calculations.

Other issues such as jitter, sporadic tasks and bursts, semaphores, etc. have also been analyzed. For further overview readings on static and dynamic priorities, we refer to Fidge [34] and Buttazzo [17]. Detailed information on EDF is available from Jeffay's thesis [51] and the *EDF Book* [111] that also discusses several variants of EDF such as *slack-based* and *least laxity first* scheduling. Complexity issues can be found in compact form in [9]. Other work provides interesting comparisons between rate-monotonic and EDF [18, 136].

### 2.1.4 Time-Driven Scheduling

Priority-driven scheduling is proven optimal and efficient under many circumstances. However, it usually results in a dynamic system behavior at run time, especially if incomplete task sets are considered. In contrast, time-driven scheduling assigns more or less fixed time slots to tasks. Each task can occupy the processor (or other resource such as a bus) for the given amount of time. When the time-slot expires, the next task is given access to the resource. There are two popular variants of time-driven scheduling: *time-division multiple access* (TDMA) and *round robin* (RR).

#### 2.1.4.1 Time-Division Multiple Access

In TDMA, time slots are assigned to tasks regardless if the tasks are active, i. e. request the resource, or not. This results in a fully static task behavior. The timing of one task does no longer depend on the behavior of any other task, and the scheduling can be easily predicted. The worst-case response time of a task $\mathcal{T}_i$ with period $T_i$, time slot $s_i$, and overall TDMA turn-time $t$ is given by:

$$R_i = \underbrace{C_i}_{\text{own core exe. time}} + \underbrace{(t - s_i)}_{\text{other slot times}} \times \underbrace{\left\lceil \frac{C_i}{s_i} \right\rceil}_{\text{max. number of slots required}} \tag{2.9}$$

In case of task recurrence, Lehoczky's windowing technique [69] for analyzing arbitrary deadlines can be applied with minor modifications. The scheduling analysis for TDMA is far less complex than the priority driven scheduling strategies, and there are considerably less publications on this topic. Furthermore, TDMA is less popular for task scheduling on processors but often used for communication scheduling (bus arbitration). An extensive overview on TDMA communication scheduling and its commercial application in the time-triggered protocol (TTP) [125, 64] can be found in Kopetz' book [65]. Other

industrial TDMA solutions can be found in the time-triggered controller area network protocol [25] (TTCAN) and in FlexRay [36].

The static nature of TDMA, although very welcome when analysis is concerned, can result in substantial inefficiencies that can become disabling as systems grow in size. Unused time slots cannot be assigned to waiting tasks, hence the system might often idle, even under heavy load.

#### 2.1.4.2 Round Robin

Round robin avoids this inefficiency by *releasing* a time slot if not required. This avoids unnecessary idle times and results in a compact schedule. The Token Ring (IEEE 802.5) communication protocol [50] with its "early token release" is an example for a round robin system.

Round robin systems perform much better than TDMA on average, but have TDMA performance in the worst-case situation. Therefore, the worst-case load and response time calculations can be re-used from TDMA. A comparison between TDMA and RR performance with a large set of experiments can be found in [20].

### 2.1.5 Other Scheduling Strategies

Cyclic or *cyclo-static* [31] scheduling is popular for signal processing applications. They have proved to be very efficient for synchronous data-flow [68] (SDF) applications [68, 66], a popular class of signal processing applications. However, such applications need not necessarily always be scheduled statically [75] but can be optimized using EDF scheduling [145]. Mok [78] has exploited cyclo-static task behavior to reduce response times in dynamic systems. First-come first-served (FCFS) –also known as first-in-first-out (FIFO)– scheduling has also been analyzed [139].

In general, countless scheduling strategies exist, including layered and hierarchical scheduling and nested run-time systems, e. g. in Flexray [36] with priority-driven scheduling "inside" TDMA; and every variation might require the existing analytical models and methods to be re-thought and adjusted based on core execution times, activation models, etc.. This thesis is not intended to thoroughly explain all existing work on scheduling and its analysis, and at this point we will stop the survey on single-processor or single bus scheduling and analysis.

### 2.1.6 Industrial Application

Nearly all of the aforementioned, and many more, scheduling strategies can be found in commercial systems. We have already highlighted the role of embedded software and operating systems and the increasing importance of networks and protocols in the introduction. Likewise, key analytical contributions

such as rate-monotonic analysis, release jitter analysis, response time analysis for arbitrary deadlines, time-slot optimization, etc. have been commercially adopted –as long as they can be efficiently automated, i. e. they are not NP-complete or NP-hard.

Tools such as TriPacific's *RapidRMA* [124], Livedevices' *Real-Time Architect* [33], *Comet* [129] from VaST, and Vector's *CANAlyzer* [130] target processor or bus scheduling analysis. They re-use core formal methods and extend them to capture specific operating system characteristics such as context switch time and OS primitives. In our own projects [16, 56], we have analyzed the *ERCOSEK* [32] automotive operating system from ETAS and could successfully extend the rate-monotonic approach to capture the complex priority and OS function structure. The *RTEMS* [79] operating system has also been analyzed [26].

These tools target a single component –processor or bus. The *TTP Software Development Suite* [126] from TTTech goes a little further and supports the design and dimensioning of multi-controller networks based on the time-division multiple access paradigm. This is eased by the static nature of such time-triggered systems, where dependencies such as scheduling anomalies are rare.

## 2.2 System-Level Extensions

Compared to the countless contributions on single-processor scheduling analysis, there are considerably less proposals for how these techniques can be extended to analyze larger systems where distributed tasks communicate with, and depend on each other, as we mentioned in the introduction. The key is to capture these dependencies in the analytical model, which can be complex and possibly leads to virtually unsolvable equations. But it is essentially needed to safely resolve the additional pitfalls such as the scheduling anomalies [37] of Section 1.1.

### 2.2.1 Homogeneous Multi-Processors

The class of *homogeneous shared-memory multiprocessors* is a straight-forward extension of single-processor systems. Task communication is not explicit but implicit through shared memory, a communication that can be modeled as shared resource access (see Section 2.1.2.4). Figure 2.3 highlights a shared-memory multiprocessor sub-system to illustrate the scope of these approaches.

The *generalized rate-monotonic scheduling* (GRMS) theory [63, 74] provides key ideas for multiprocessor extensions [106]. Similar extensions are known for EDF [111, 51] and other schedulers. Fundamentals of multiprocessor synchronization can be found in [91].

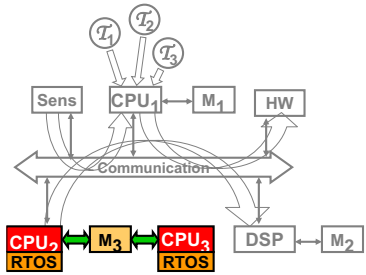



*Figure 2.3.* Scope of Homogeneous Shared-Memory Multiprocessor Analysis

More complex systems exhibit, in addition to the resource conflicts due to shared-memory access, precedence dependencies where one task "waits" for the data from another one. Fully cyclic executives, i. e. systems that apply *periodic (cyclic) scheduling* regardless of the actual communication dynamics, behave like pipelines and ensure proper precedences. Blazewicz and Jeffey derived local task deadlines from end-to-end constraints [11, 51] under EDF scheduling. Mathur, Dasdan, and Gupta [76, 28] introduced a rate-based approach that calculates maximum task frequencies for cyclic tasks. However, the necessity of fully periodic schedules is an overly constraining requirement [6]. It leads to static system behavior and considerably reduces system utilization and efficiency.

### 2.2.2 Holistic Schedulability Analysis

Audsley et. al. [6] recognized that static cyclic scheduling is not sufficient for the requirements of dynamic, reactive embedded real-time systems, and they identified dynamic scheduling under precedence constraints to be a major challenge to multiprocessor scheduling analysis.

Tindell [120] introduced the notion of time-offsets of tasks, which were already known not to be easy to handle (see Section 2.1.2.5), to model task delays that result from precedences. The response time of one task (the predecessor) becomes the time-offset of another task (the successor), ensuring that the precedence is fulfilled without compromising dynamic scheduling and analysis. This way, Tindell and Clark [122] could establish, in addition to the known response time calculations, timing equations for all dependencies in the system. The result is an equation set that entirely captures system timing with all its dependencies, and they called their approach *holistic schedulability analysis* [122]. The approach was first applied to a system of distributed tasks, preemptively scheduled under fixed priorities, communicating over a TDMA

bus. Later, they also investigated the automotive, priority-based CAN (controller area network) bus [123].

The idea was adopted by other researchers. Gutierrez, Palencia, and Harbour extended Tindell's idea by allowing *dynamic offsets* [82] to consider variations in task response times. They also provided an enhanced *best-case response time* analysis [40, 81] to improve worst-case schedulability analysis, a topic strongly related to the problem with scheduling anomalies [37]. Their basic holistic analysis approach [41, 83] has been extended to support complex task communication and synchronization mechanisms [42]. Later, they applied similar ideas to EDF scheduling [84]. The "Modeling and Analysis Suite for Real-Time Applications" (MAST) [45, 128] can be configured to analyze different system set-ups.

Because of the importance of tight dynamic offset data, an increasing number of researchers is investigating best-case scheduling and, in particular, response time analysis for single processors [62, 47, 92, 49]. However, the number of groups focusing on holistic techniques for distributed systems is still relatively small.

Yen and Wolf [137, 138] used virtual delay tasks to model offsets in static priority scheduling. Eles and Pop –like Tindell and Clark– analyzed systems with priority-driven task scheduling combined with a TDMA bus protocol [87, 29] and derived heuristics for bus access optimizations. Later, they extended their ideas to also support hybrid communication protocols with static (time-triggered) and dynamic (event-triggered) messages [88]. This allows configuration of the system from fully static (for instance using TTP [125]) to mixed (Flexray bus [36]) to highly dynamic (CAN [85]) systems, providing a more structured view on the individual influences. They also accounted for conditional task dependencies and task synchronization [30]. Pop's thesis [86] provides a detailed overview.

The holistic approach, as a straight-forward extension of single-processor scheduling analysis with offsets, appears as an intuitive procedure to tackle multiprocessor scheduling. Unfortunately, many practically relevant scheduling analysis problems have recursive solutions and have been proven NP-hard even for seemingly simple system configurations [73, 71]; and the complexity of the holistic analysis is even higher [10, 9]. Often, the whole equation set with manifold dependencies needs to be iteratively solved. To manage complexity, all mentioned groups propose approximations and furthermore restrict themselves to homogeneous task scheduling, either with fixed priorities or EDF. Communication is mostly considered static using TDMA protocols which apparently reduces dynamic dependencies and substantially simplifies the system timing equations. Finally, complex applications with e. g. cyclic dependencies are mostly not discussed, this would introduce just another analysis iteration.

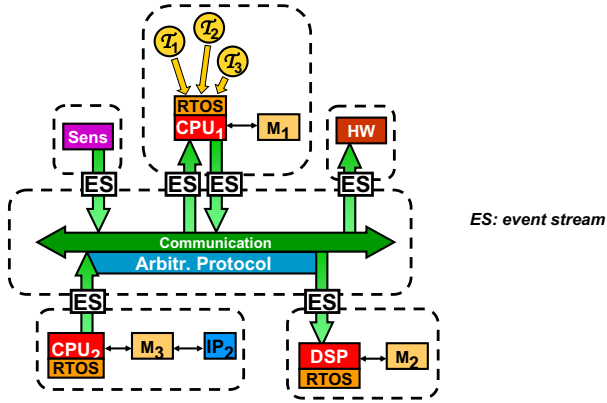



*Figure 2.4.* Scope of Homogeneous Flow-Based Scheduling Analysis

Despite these weaknesses, the holistic approach appears as an efficient and practically useful technology for special classes of distributed systems, such as automotive, where fixed priority tasks and periodic communications are current practice. The Volcano tool chain [132] applies the holistic approach to automotive applications and analyses the priority-driven CAN communication at the system level. Interestingly, Volcano –as TTTech does– requires their own, proprietary bus access hardware and software drivers to be used with their approach. This specifically hinders the integration of legacy or other components, which is considered a serious disadvantage in industry.

## 2.3 Flow-Based Analysis

Even before Tindell's holistic approach appeared, another approach to system-level scheduling analysis was published. Gresser [38, 39] viewed tasks as independently scheduled and *locally analyzable entities* that communicate or interact *via event streams*. The scope of the flow-based approach is illustrated in Figure 2.4. The event streams (ES) represent interfaces between the individual components and sub-systems. This gives structure, not only to the involved equations, but also enables a structured solution procedure, which is considered a major advantage over the monolithic, unwieldy equations of holistic analysis.

### 2.3.1 Event Vectors

Gresser introduced *event vectors* to model task activation for event-driven EDF scheduling. The model is very powerful and can capture periodic events,

jitter, and burst in a uniform manner. As a key contribution, his local EDF scheduling analysis does not only provide task response times but also the corresponding *output event streams* using the same event vector notation. In this way, the output of one task can be directly used for activation modeling and analyzing a dependent task, because task output and task activation use the same model. In other words, events are seen as traveling or flowing through the network of tasks and communications and activate the corresponding actions. This turns system-level scheduling into a *flow-analysis problem* that can be iteratively solved.

Mathematically speaking, Gresser also creates a set of equations to capture system timing, just like the holistic models. However, Gresser's approach yields a *structured* timing model. Each component has its own, relatively comprehensible set of equations for local scheduling analysis. At the system level, these component equations are combined into a system-level model through the input/output specifications of the tasks, based on the event vectors. The analysis of the system benefits from this structure. Starting at system inputs, analysis follows the flow of events and *locally* analyzes one component after another, until all components have been successfully analyzed, or a constraint is violated.

### 2.3.2 Arrival Curves

Thiele et. al. [116] defined another flow model specifically targeting network processor design. Where Gresser defined event vectors, Thiele et. al. use *arrival curves* to capture incoming (and outgoing) requests. And instead of suitable standard scheduling equations, they propose to use *service curves* for modeling resource availability, and apply a newly developed algebra, called *real-time calculus* [116], to establish relations between input and output curves of a component. The basic ideas have roots in *network calculus* [27, 15, 14].

They started using continuous arrival curves with numerical representations [116], and the initially proposed [117] curve processing was restricted to preemptive static priority scheduling [22]. They subsequently extended their model to also account for non-preemptive packet processing, as found in real-world network processors, and to support other scheduling strategies. But arbitrary curves quickly appeared insufficient for key analytical observations, and they resorted to approximation models to bound arbitrary curves [23]. Chakraborty's thesis [21] provides an adequate overview.

From a system-level perspective, both Gresser and Thiele define an event model and propagate event streams through locally analyzable components in a global iterative analysis. The clear separation of a) local component analysis and b) component coupling through event streams breaks down the complexity of the overall problem and gives confidence to analysts and designers. The

iterative way of analysis enabled by the event stream view is also a suitable way to understand the complex dependencies that distributed systems exhibit.

## 2.4 Previous Own Work

In our own research projects, we have investigated general system models for the verification of non-functional requirements, specifically with respect to scheduling [93]. Our SPI (System Property Intervals) model [141, 144] was introduced as an *intermediate design representation* to interface between existing models of computation and analytical task models. A particular goal of the SPI project was to reach a reasonable level of abstraction that allows a *homogeneous view* of *heterogeneously specified systems*, e. g state machines coupled with data-flow graphs. We have shown how specification languages with different models of computation such as the state-based SDL [103] and the data-flow based Matlab/Simulink [115] can be transformed into the SPI model [53, 59], and how timing constraints can be captured using event models [57].

Much of this work was carried out in close co-operation with Thiele's group at the Swiss Federal Institute of Technology, Zurich. This group had a similar model called FunState [119, 118, 112], and basic concepts, such as the representation of operational modes [143] and function variants [99], were developed together. For overview readings we refer to [144], while Ziegenbein's thesis [140] provides a detailed introduction into SPI.

Another project, SymTA/P (Symbolic Timing Analysis for Processes) targets running time analysis of single tasks [134, 135] by program path analysis and architecture modeling with many optimizations. An extensive summary can be found in Wolf's thesis [133]. First ideas of breaking down the complexity of SPI into smaller sub-problems, that can be combined together with SymTA/P and the aforementioned work on scheduling analysis, were proposed in [142]. We have investigated the role of scheduling analysis in flexibility analysis [96, 100, 46]. Finally, the SymTA/S project brought together key ideas of both projects SPI and SymTA/P [146]. This thesis presents the basic approach underlying SymTA/S, while extensions are surveyed in the summary.

## 2.5 Summary & New Approach

This summary will conclude the review on existing techniques with an evaluation of their applicability in the area of heterogeneous, distributed embedded systems, and with respect to the thesis objectives defined in the introduction.

### 2.5.1 Evaluation

We have seen that the *single-processor models* (see Section 2.1) such as rate-monotonic scheduling are comparably easy to set-up, to understand, and

to analyze. The number of industrial applications that utilize such technologies shows its importance in real-time system design as well as its acceptance in industry. However, these techniques have a significant disadvantage, they are limited to a single processor or bus. There are in fact extensions that target *multiprocessors* (Section 2.2.1), which re-use much of the established single-processor scheduling models which is considered a major advantage. Most of these extensions, however, require cyclic schedules which decouples the individual tasks from each other and simplifies the analysis. Such systems suffer from a reduced efficiency, and these techniques do not represent a relevant solution for the design of competitive embedded systems.

The *holistic approach* (Section 2.2.2), in principle, captures highly dynamic interactions between tasks on different processors and covers more efficient systems that are not restricted to having periodic, cyclic schedules. In practice, however, the approach often suffers from its complexity in general. The resulting recursive system level equation set is not easily understandable which counters the demand for simple models that designers can oversee.

There are few specialized classes of applications, such as those which use TDMA-based communications, for which efficient holistic models exist. The cyclic nature of TDMA decouples the task dependencies to a reasonable degree and simplifies the equation set and its solutions. However such models lack the flexibility to support the "cut&paste" design style. Exchanging components is complex and usually requires the derivation of a new equation set. This modeling complexity seems one of the main reasons why holistic models have so far been only developed for specialized classes of architectures. This is also reflected in the MAST tool [45, 128] which requires a specific holistic analysis algorithm for each system class.

The *flow-based approaches* of Section 2.3, with their clear separation into local scheduling analysis and the global event stream flow, break down the complexity and provide a structured and comprehensible view of system-level scheduling. Comprehensibility is an important requirement for designers. Another key advantage over the holistic approaches is the higher flexibility with respect to system size and heterogeneity. Thiele et. al. [116] can include static or dynamic priority scheduling with preemptive or non-preemptive behavior. Although Gresser [38, 39] only demonstrated his approach for EDF scheduling, one could theoretically include *any* sub-system scheduling strategy for which an analysis exists that supports the flow models. This provides the necessary modularity to support "cut&paste" system integration. We summarize that the flow-based approaches appear most promising for handling the complex system-level scheduling problems of today's system.

The flexibility, however, has its limitations. Thiele and Gresser define a novel and very generalized flow model for their approaches that must be *homogeneously* used during system analysis. Although these general models the-

oretically include the standard models from real-time systems research, none of the existing analytical contributions from Section 2.1 is directly supported because they use more constrained models. RMS (rate-monotonic scheduling) for instance knows of periods and deadlines but has no notion of event vectors nor arrival curves and can, hence, not be re-used.

### 2.5.2 Revised Objectives & Basic Idea

After this evaluation we shall now rephrase and particularize the objectives of this thesis. There are three key requirements:

1 We are seeking an approach that meets current industrial design practice, i. e. the approach shall provide the *flexibility* and *scalability* of the heterogeneous "cut&paste" design style, different system parts need be independently *configurable*.

2 We want to *re-use* as much work on local component scheduling analysis as possible. This allows the *integration of established techniques and tools* for sub-systems and increases system understanding and the acceptance of the approach.

3 The system-level analysis must be *comprehensible* (enough) so that designers can follow the underlying principles. Complex mathematics, models, and algorithms are extremely unlikely to attract designers in industry and must be avoided.

Basically, meeting the above requirements means combining the advantages of the existing work in a reasonable way. We want to re-use the local techniques from Section 2.1 for sub-system and component analysis. This makes the details of the analysis understandable and allows designers to (re-)use established local techniques and models. Then, we will combine the individual models using the flow-based approach (Sec. 2.3). The event stream view enables an analysis procedure that follows the system-level dependencies that scheduling introduces in an understandable way. It also provides the flexibility and scalability that todays design style requires.

### 2.5.3 Key Challenges

A key problem with this idea is the *incompatibility of the local input and output models*. The existing flow-based approaches define a *single event model* for component and task inputs and outputs to be *homogeneously* used. But the host of known local techniques uses a big variety of input activation models. We summarized a few in Section 2.1.2.3. In order to re-use those techniques, the flow-based framework must support a *heterogeneous mixture* of event models.

These models are fundamentally different from the event vectors or arrival curves, and are also incompatible with each other. Furthermore, output event

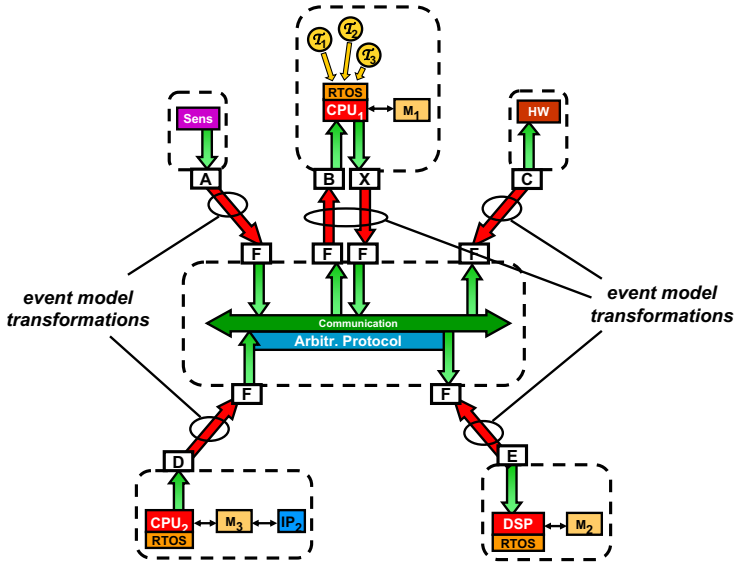



*Figure 2.5.* Heterogeneous Flow-Based Scheduling Analysis

models have hardly been investigated in literature so far. These are –as will be shown in Chapter 4– usually more complex than input models and increase the model diversity further. Figure 2.5 shows the example system split into 5 computation sub-systems and the integration bus. Each component brings a different event model as its interface to other components (A..X). The $\mathsf{CPU}_1$ sub-system even has two different models at input (B) and output (X). This illustrates the key challenge: finding a solution to the event model incompatibility problem in system-level scheduling analysis.

This thesis describes a way to overcome this event model incompatibility. We will introduce *Event Model Interfaces* and *Event Adaptation Functions* to transform one model into another, as indicated in Figure 2.5. The importance of transformations between different event models has been widely neglected in literature. In general, global heterogeneous system analysis using standard techniques is currently not possible since the respective local analysis techniques use incompatible event models.

The event model transformations essentially provide the missing link between the local (re-)use of known models and the flow-based global analysis procedure that relies on common model interfaces. Developing such event

model transformations, together with a system-level analysis procedure that utilizes heterogeneous event models are the main contributions of this thesis.

### 2.5.4 Detailed Outline

In the next Chapter, we survey once again the existing local scheduling and analysis approaches, but this time with special focus on the used input or task activation models. The vast majority of techniques share few common models. The way specific model properties are used and exploited in the analysis is also rather similar. This helps to classify models and to provide the necessary parameter extractions for the event model transformation step.

No less important for this step is the knowledge of the task output event streams that is analyzed in Chapter 4. Interestingly, output event streams and models have received far less attention than input models, most probably because their importance for system-level analysis had not been demonstrated before. Scheduling and task execution have a remarkable influence on output timing and can lead to complex event streams. We will see that not all output streams can be covered by parameters used for input activation modeling. Therefore, we marginally extend the known input event models without significantly complicating them. In contrast, a new self-contained and structured event model set is proposed that can be consistently used for input and output modeling. As an efficient compromise between model simplicity and completeness, we call these models *standard event models*.

After the basic input and output models have been defined, the next two chapters are concerned with the actual model incompatibility. In Chapter 5, we first of all investigate those situations where mathematical parameter transformations are sufficient to transform one model into another without changing the actual stream timing. This is usually the case if the target model of the transformation is more general than the source model. We call such transformations *Event Model Interfaces*.

Transformations from a more general into a more constrained model are analyzed in Chapter 6. They are considerably more complex, since they require key properties of the stream timing to be changed. So called *Event Adaptation Functions* apply automatic traffic shaping to match the required properties. We also explain how designers can manually control the shaping process and optimize the system. Shaping is very popular and powerful in the area of network optimization, and it can be elegantly included in the new approach.

Chapter 7 brings together all concepts and presents our novel compositional system-level scheduling analysis procedure, based on standard event models and standard sub-system techniques. A set of experiments is carried out, which explain the analysis procedure step by step. We specifically discuss complexity issues and propose reasonable solutions for cyclic dependencies, that often

turn the iterative flow analysis into a convergence problem. Such cyclic dependencies in dynamic systems have hardly been analyzed before.

In Chapter 8, we conclude this thesis with a summary, and identify and evaluate different possible extensions for further research.

Chapter 3

# INPUT EVENT MODELS

This chapter reviews in detail the four most popular event models from the literature on real-time systems. The discussion specifically identifies the common properties of event models and provides characteristic functions for these properties.

## 3.1 Task Activation and Event Streams

*Local scheduling analysis techniques typically use event models to capture incoming load.*

We have already used this statement repeatedly in the previous chapters. It is, however, not always fully obvious that the statement is true. In many publications, the model definitions are implicit, i. e. they are part of, or they result from, some other definitions. Rate-monotonic scheduling, for instance, does not explicitly define a model of periodic events, the whole theory just assumes periodically executed tasks.

Furthermore, not all publications treat task activation as "incoming load". Again in RMS theory, tasks are assumed to execute periodically but it is not specified how they are activated. In practice, periodic task execution is achieved using a system timer that periodically emits interrupt requests. These timer interrupts define an event stream that activates the task. Such activation is usually considered *time-triggered*. Periodic execution is often found in relatively deterministic (or static) systems, but also has its application in sporadic and asynchronous environments where periodic polling is a popular mechanism to obtain deterministic scheduling.

In other situations, a task such as a communication driver is not activated by a timer event but upon the reception of a protocol frame from an external bus. More precisely, a bus-controller hardware block is first responsible for collecting the bits and bytes from the bus wires. Then, it signals the reception of

the whole frame to the driver task, again, using an appropriate interrupt. Such task activation is usually considered *event-triggered*. Driver tasks and other "sporadic servers" [108] are often found in dynamic systems which operate in non-deterministic environments.

Other tasks may be activated upon some timeout event relative to another event, and direct task activation is also possible. However, *task activation can always be modeled by events*, regardless whether time-triggered or event-triggered, periodic or sporadic. This way, *tasks are always reactive*, they react to their activating event. Just the source of the event can vary from very deterministic system timers to highly dynamic environmental requests.

This illustrates the task model used in this thesis. Tasks are activated by event streams; and each task consumes one event per activation. Hence, the stream of activating events is considered an *input event stream*, providing information about the *incoming load*. Similar observations will be presented for output event streams in Chapter 4, and a distinction between activating inputs and output generation provides a reasonable formal foundation for the event model interfaces in Chapter 5.

There are four input event models, which are of major importance in real-time scheduling theory:

- strictly periodic events,
- periodic events with jitter,
- sporadic events, and
- sporadically periodic events, often called "sporadic bursts".

These four event models are sufficient to cover the vast majority of popular local scheduling and analysis techniques, such as introduced in Section 2.1.

## 3.2 Common Properties of Event Streams

In order to compare different event streams, we need to capture their *common properties*; common in the sense that they apply to all four event models mentioned (and others). These properties lay the foundation for all later compatibility tests, and they permit the derivation of event model interfaces and adaptations.

All scheduling analysis strategies exploit two key properties of an incoming event stream. First, they require the *number of events* that arrive within a given interval of time to be known. With this information, the analysis algorithms predict, for instance, the number of task activations, or the number of preemptions by a higher-priority task. Second, all analysis techniques require a *minimum temporal separation* (or distance) between a given number of successive events in a stream to be known. This is used to predict future event arrival times, required to derive deadlines or determine busy windows.

**Definition 3.1.** ***(Event Stream)***
*An Event Stream $\mathcal{S}$ is a numerical representation of the possible timing of event occurrences (or event arrivals). An event stream is defined by four characteristic functions:*

$$\eta^+ \;:\; \mathbb{R}^+ \mapsto \mathbb{N}^+ \,, \tag{3.1}$$
$$\eta^- \;:\; \mathbb{R}^+ \mapsto \mathbb{N}^+ \,, \tag{3.2}$$
$$\delta^- \;:\; \mathbb{N}^+ \setminus \{0,1\} \mapsto \mathbb{R}^+ \text{ , and} \tag{3.3}$$
$$\delta^+ \;:\; \mathbb{N}^+ \setminus \{0,1\} \mapsto \mathbb{R}^+ \,. \tag{3.4}$$

*The function $\eta^+(\Delta t)$ returns the maximum number of possible event occurrences within a time interval of size $\Delta t$[1]. Likewise, the function $\eta^-(\Delta t)$ returns the minimum number of event occurrences in that interval $\Delta t$.*

*The functions $\delta^-(n)$ and $\delta^+(n)$ return the minimum and maximum distance between $n$ successive events in the stream.* ■

These four characteristic functions are further used in the actual analysis algorithms. We start the survey with strictly periodic events.

## 3.3 Strictly Periodic Events

Strictly periodic event streams are characterized by one single parameter, the period $T$, which must be a positive and non-zero real. The characteristic functions are defined based on this parameter.

### 3.3.1 The $\eta(\Delta t)$ Functions

Very often, the calculations of $\eta^+(\Delta t)$ and $\eta^-(\Delta t)$ are not explicit but hidden in the overall theory. Liu and Layland [73], for instance, provided a sufficient schedulability test based on the system utilization $U$ as the sum of the utilization portions of all tasks:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} < n(2^{\frac{1}{n}} - 1) \tag{3.5}$$

$C_i$ represents the *worst-case execution time* of task $\mathcal{T}_i$. This is divided by the task's period $T_i$ to determine the task's long-term average load in percent of the avaliable processor performance. As part of a more complex but exact schedulability test [73], they calculate the worst-case workload within an interval of $\Delta t$:

$$w(\Delta t) = \sum_{i=1}^{n} \left( \left\lceil \frac{\Delta t}{T_i} \right\rceil C_i \right) \tag{3.6}$$

[1]The set $\mathbb{R}^+$ contains all non-negative reals, including zero. Likewise, $\mathbb{N}^+$ contains all non-negative integers, including zero.

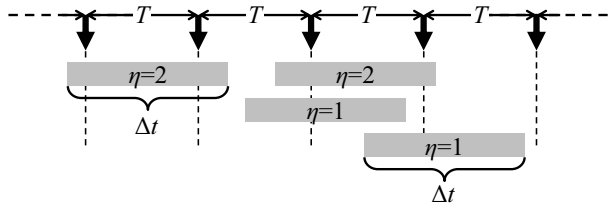



*Figure 3.1.* Time Intervals and Number of Events of Periodic Event Streams

The worst-case response time ($R_i$) approach of Joseph and Pandya [60] contains similar terms:

$$R_i = C_i + \sum_{j \in \mathrm{HP}(i)} \left( \left\lceil \frac{R_i}{T_j} \right\rceil C_j \right) < D_i = T_i \tag{3.7}$$

In both equations 3.6 and 3.7, the rounded term calculates the *maximum number* of task activations within a given amount of time. And the subsequent worst-case calculations further use this term. Basically, the maximum number $\eta_{\mathsf{P}}^{+}$ of periodic events[2] ($\mathsf{P}$) within a given amount of time $\Delta t$ is given by:

$$\forall\ \Delta t > 0 : \eta_{\mathsf{P}}^{+}(\Delta t) = \left\lceil \frac{\Delta t}{T} \right\rceil \tag{3.8}$$

The number of events that can arrive within some time $\Delta t$ is one of the key properties used in scheduling and schedulability analysis techniques, not only periodic or rate-monotonic.

A more general way to approach this function is to first think of continuous functions. Figure 3.1 shows a periodic event stream and several intervals of size $\Delta t$. Obviously the number of events within an interval can vary and depend on the absolute position of the interval within an event stream. We can easily determine the *average* number of periodic of events $\tilde{\eta}$ within a given $\Delta t$:

$$\tilde{\eta}_{\mathsf{P}}(\Delta t) = \frac{\Delta t}{T} \tag{3.9}$$

**Upper-Bound Arrival Function.** When the continuous function is known, the discrete maximum numbers are then obtained by appropriate rounding. With respect to Figure 3.1, rounding corresponds to aligning one of the interval bounds to event arrival times. Time is considered to increase to the

[2] We will use the index $\mathsf{P}$ to specify *periodic* event streams, and later event models

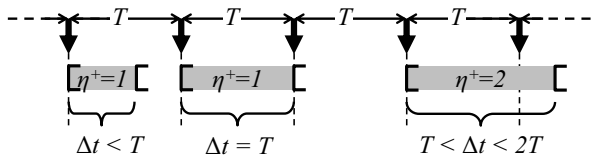



*Figure 3.2.* Maximum Number of Events of Periodic Event Streams

right. Figure 3.2 shows three *left-closed right-open* time intervals that are *left-aligned* to events. The combination of left-side alignment and left-closeness ensures maximum values for $\eta$. Practically speaking, the first event is just included in the interval (at the left bound), and all others arrive as soon as possible. No other interval would yield a higher number of events. This property has already been exploited by Liu and Layland in their seminal paper on scheduling analysis [73].

Basically, the upper-bound $\eta^+(\Delta t)$ function is obtained by *rounding up* the average number as calculated by Equation 3.9. This function was already provided by Equation 3.8:

$$\forall\, \Delta t > 0 : \eta^+_{\mathsf{P}}(\Delta t) = \lceil \tilde{\eta}_{\mathsf{P}}(\Delta t) \rceil = \left\lceil \frac{\Delta t}{T} \right\rceil$$

The restriction to $\Delta t > 0$ has an intuitive practical reason. The term on the right hand side of the equation is commonly found in many response time algorithms to calculate the number of preemptions during the *non-zero* execution time of a periodic task. Hence, $\Delta t = 0$ is not practically important. For the formal issues later in this thesis, we define the function value to be zero (0) for $\Delta t = 0$. This is also consistent with the mentioned half-openness of the interval. A half-open interval of width zero represents an empty set. Hence, no event occurrence time can be included. A general reasoning about how to best model and capture time and its influences on computer systems can be found in [102].

The $\eta(\Delta t)$ functions allow *event arrival curves* to be drawn, which are known from other work [27] . The maximum function $\eta^+(\Delta t)$ leads to the *upper bound* arrival curve, shown in Figure 3.3. Specifically the left closeness lets the interval just "catch" the first event. The actual *as-soon-as possible* event arrival scenario, which is illustrated using the arrows below the figure, illustrates the correspondence between an arrival curve and the relative timing of events.

**Lower-Bound Arrival Function.** While an upper bound is sufficient for single processor system analysis, distributed systems exhibit scheduling

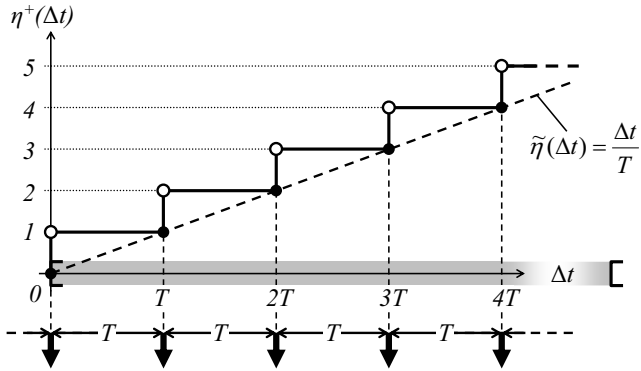



*Figure 3.3.* Upper Bound Arrival Curve of Strictly Periodic Events

anomalies [37] which can turn subsystem best-case into system worst-case behavior, and vice versa. Therefore, also the best-case, i. e. the situation with minimum load, needs to be known. In order to calculate such best-case situations, the minimum number of events within a given interval of time needs to be known.

The calculations are similar to those that yield the maximum number. But instead of left-closed intervals, we need to consider right-closed intervals, since the event to which the interval is aligned must just be excluded (just "missed") from the interval. So the first event that is actually included in the right-closed intervals arrives after one full period $T$. All later events arrive strictly periodically.

These observations lead to the lower bound arrival curve shown in Figure 3.4. Specifically the right-closed half-openness lets the interval miss the first event. The curve straight-forwardly yields the actual function for the minimum number of events during time interval $\Delta t$, given by Equation 3.10. Again, we round the $\tilde{\eta}$ function, but this time we round down since we are looking for minima.

$$\forall\, \Delta t > 0 : \eta_{\mathsf{P}}^{-}(\Delta t) = \lfloor \tilde{\eta}_{\mathsf{P}}(\Delta t) \rfloor = \left\lfloor \frac{\Delta t}{T} \right\rfloor \tag{3.10}$$

Figure 3.5 shows both the upper- and the lower-bound arrival curves. The actual values of $\eta(\Delta t)$ can vary between these two curves and is illustrated by the gray area. Mathematically, this is captured by the following equation:

$$\eta_{\mathsf{P}}^{-}(\Delta t) \leq \eta_{\mathsf{P}}(\Delta t) \leq \eta_{\mathsf{P}}^{+}(\Delta t) \tag{3.11}$$

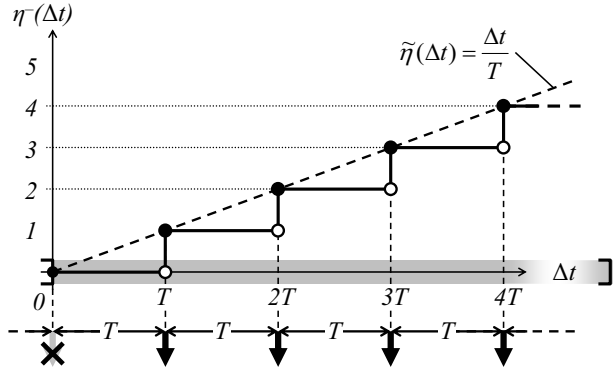



*Figure 3.4.* Lower Bound Arrival Curve of Strictly Periodic Events

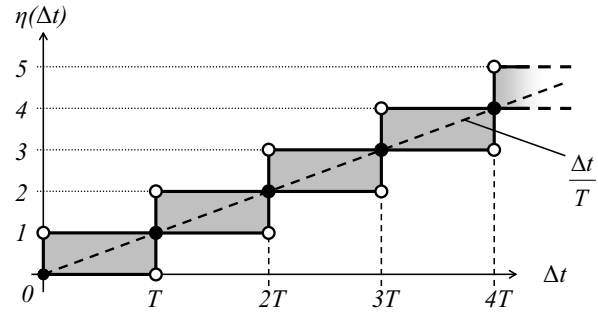



*Figure 3.5.* Both Arrival Curves of Strictly Periodic Events

The uncertainty, i. e. the difference between both bounds, can be captured by the *uncertainty interval* $\eta^{\mathrm{I}}(\Delta t)$:

$$\eta_{\mathsf{P}}(\Delta t) \in \eta^{\mathrm{I}}_{\mathsf{P}}(\Delta t) = \left[\eta^{-}_{\mathsf{P}}(\Delta t); \eta^{+}_{\mathsf{P}}(\Delta t)\right] \tag{3.12}$$

### 3.3.2 The $\delta(n)$ Functions

As already mentioned in Section 3.2, there is a second function of central importance. The $\delta(n)$ functions returns the distance $\Delta t$ between $n$ (at least two) successive events in the stream. There is a minimum and a maximum value of that function, distinguished by $\delta^{-}(n)$ and $\delta^{+}(p)$. Roughly speaking,

the $\delta(n)$ functions are the inverse of the $\eta(\Delta t)$ functions. The key difference is that the $\eta(\Delta t)$ functions start at $\Delta t = 0$, while $\delta(n)$ is not defined –and obviously makes no sense– for $n < 2$.

It is intuitive that the distance between two events in a periodic stream always (minimum and maximum) equals the period $T$, the distance between three events is twice the period, and so on, resulting in Equation 3.13.

$$\forall\, n \in \mathbb{N} \setminus \{0,1\} : \delta^-_{\mathsf{P}}(n) = \delta^+_{\mathsf{P}}(n) = (n-1)T \tag{3.13}$$

Many analysis techniques require a *minimum temporal separation* (or distance) of two events ($\delta^-(2)$) in order to check for task recurrence. In Equation 3.7, this information is used to formulate a deadline for task $\mathcal{T}_i$. In periodic events, the distance between any two events simply equals the period $T$, and the tasks deadline is set to this distance. Analysis techniques for other event streams apply other deadlines to guarantee non-recurring tasks. These deadlines can be derived from the event stream's characteristic functions. In Section 2.1.2.2, we mentioned the *windowing technique* of Lehoczky [69] and Tindell [121]. They calculated the size of the *busy window* $w_i(q)$ for a given number $q$ of invocations of task $\mathcal{T}_i$ using Equation 2.3. In order to determine the actual response time of the task, the invocation time of the $q$th invocation relative to the start of the first invocation is subtracted from $w(q)$. This invocation time equals the distance between $q$ successive events (see Equations 2.4 and 2.5):

$$R_i = \max_{q=1,2,\ldots} \big(w_i(q) - \underbrace{(q-1)T_i}_{\delta_i^-(q)}\big) \tag{3.14}$$

Equation 3.14 is of major importance for several response time techniques for tasks with arbitrary deadlines, and it applies to other event streams, as well. A detailed explanation of the windowing technique and busy periods can be found in [69, 121, 122].

## 3.4 Event Streams vs. Event Models

We have seen that strictly periodic streams share key common properties. They are characterized by a single parameter, the period $T$, and all scheduling analysis techniques that are based on strictly periodic event streams exploit the given characteristic functions. These commonalities are captured by an *event model*.

**Definition 3.2.** ***(Event Model)***
*An Event Model $\mathcal{EM}$ is a set of event streams that share common qualitative properties. An Event Model defines:*

*a) a parameter tuple $\pi_{\mathcal{EM}}$ that captures the common properties of the streams. One parameter tuple is associated with each stream in the set. We define $\Pi_{\mathcal{EM}}$ as the set of all possible parameter tuples $\pi_{\mathcal{EM}}$.*

*b) the four characteristic functions, $\eta^+_{\mathcal{EM}}(\Delta t)$, $\eta^-_{\mathcal{EM}}(\Delta t)$, $\delta^-_{\mathcal{EM}}(\Delta t)$, and $\delta^+_{\mathcal{EM}}(\Delta t)$, based on the parameter tuple $\pi_{\mathcal{EM}}$. These functions apply to all streams in the set.*

*c) a function*

$$\mathcal{S}_{\mathcal{EM}} : \Pi_{\mathcal{EM}} \mapsto \mathcal{EM} \tag{3.15}$$

*that returns an event stream with the specific parameters $\pi_{\mathcal{EM}}$ provided as the arguments to that function.*

*An event stream $\mathcal{S}_{\mathcal{EM}}$ is an* element *of an Event Model $\mathcal{EM}$, if and only if it has a valid parameter tuple $\pi \in \Pi_{\mathcal{EM}}$ and the characteristic functions associated, that the event model defines.* ■

This allows the *model of strictly periodic events* to be formally defined:

$$\begin{aligned}\mathcal{EM}_{\mathsf{P}} = \{\, \mathcal{S}_{\mathsf{P}}(\pi_{\mathsf{P}}) \quad | \quad & \pi_{\mathsf{P}} \in \Pi_{\mathsf{P}} = \{(T) | T \in \mathbb{R}^+ \setminus \{0\}\}, \\ & \eta^+_{\mathsf{P}}(\Delta t) = \left\lceil \frac{\Delta t}{T} \right\rceil, \\ & \eta^-_{\mathsf{P}}(\Delta t) = \left\lfloor \frac{\Delta t}{T} \right\rfloor, \\ & \delta^-_{\mathsf{P}}(n) = (n-1)T, \\ & \delta^+_{\mathsf{P}}(n) = (n-1)T\}\end{aligned} \tag{3.16}$$

## 3.5 Periodic Events with Jitter

The assumption of strictly periodic tasks is overly restrictive. Many periodic systems exhibit a so called *jitter* that captures the distortion a periodic stream might experience. Figure 3.6 shows a periodic event stream with jitter. The period is given by $T \in \mathbb{R}^+ \setminus \{0\}$, the jitter by $J \in \mathbb{R}^+$. Jitter adds uncertainty to the model of strictly periodic events. The events arrive periodically on average, but each individual event is allowed to deviate (indicated by the left-right arrows) with respect to a virtual reference period (gray). The jitter parameter actually bounds the maximum deviation. Many approaches to scheduling analysis with jitter assume a jitter that is less than the task period ($J < T$).

### 3.5.1 The $\eta(\Delta t)$ Functions

Figure 3.6 contains four half-open intervals of equal width. In Figure 3.1, we have seen that the maximum and minimum observable number of events for strictly periodic events differs by at most one (1). In Figure 3.6, the numbers differ by three (3), although all intervals are left-aligned to an event.

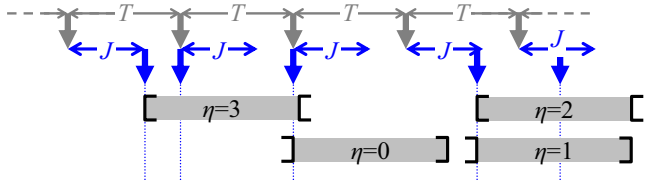



*Figure 3.6.* Time Intervals and Number of Events of Periodic Events with Jitter

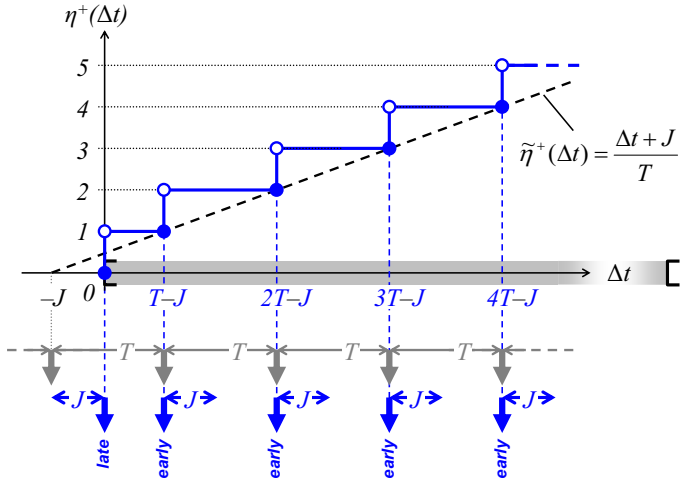



*Figure 3.7.* Upper Bound Arrival Curve of Periodic Events with Jitter

**Upper-Bound Arrival Function.** To which event shall the intervals be aligned? Again, the existing work on scheduling theory provides the answer. Jitter models were first considered by Locke [75]. He constructed a worst case event arrival scenario as follows. In a stream of events, he assumed one event to arrive *as late as possible* within the maximum allowed time deviation, and all subsequently arriving events *as early as possible* after that first event. This scenario as well as the resulting upper bound arrival curve are drawn in Figure 3.7.

In the worst-case situation, all events in the jittered stream (except the first one) re-arrive $J$ earlier than in a strictly periodic stream. More precisely, only the first event arrives later but also the time offset of $\Delta t$ is shifted by the same

amount, namely the jitter $J$. Compared to the upper bound arrival curve of the strictly periodic events (Fig. 3.3), the curve of the jitter stream has been shifted to the left by $J$, and has the following continuous $\tilde{\eta}^+(\Delta t)$ function:

$$\forall\, \Delta t > 0 : \tilde{\eta}^+_{\mathsf{P+J}}(\Delta t) = \frac{\Delta t + J}{T} \tag{3.17}$$

The actual $\eta^+$ function is given by Equation 3.18.

$$\forall\, \Delta t > 0 : \eta^+_{\mathsf{P+J}}(\Delta t) = \left\lceil \tilde{\eta}^+(\Delta t) \right\rceil = \left\lceil \frac{\Delta t + J}{T} \right\rceil \tag{3.18}$$

To demonstrate the use of such models, we briefly introduce Audsley's jitter analysis [4] that extends the approach of Joseph and Pandya [60] and captures the influence of jitter on the response time. In his model, the maximum number of events for a given time interval is used to determine the so called worst-case interference $I_i$. Audsley calculated the response time as follows:

$$R_i \leq D_i = T_i \tag{3.19}$$

$$R_i = J_i + r_i \tag{3.20}$$

$$r_i = C_i + I_i \tag{3.21}$$

$$I_i = \sum_{j \in \mathrm{HP}(i)} \Bigg( \underbrace{\left\lceil \frac{r_i + J_i}{T_i} \right\rceil}_{\eta^+_{\mathsf{P+J}}(r_i)} C_j \Bigg) \tag{3.22}$$

**Lower-Bound Arrival Function.** A situation with the minimum number is shown in Figure 3.8. The interval is right-closed half-open and aligned to an "early" event. Due to the left-openness of the interval, this event is just missed. All other events arrive "late", i. e. they experience the maximum deviation. This time, the curve has been shifted to the right and the continuous $\tilde{\eta}^-(\Delta t)$ function is:

$$\forall\, \Delta t > 0 : \tilde{\eta}^-_{\mathsf{P+J}}(\Delta t) = \frac{\Delta t - J}{T} \tag{3.23}$$

Since this function can be negative for $\Delta t$ less than the jitter $J$, the rounded value must be bounded to at least zero:

$$\forall\, \Delta t > 0 : \eta^-_{\mathsf{P+J}}(\Delta t) = \max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right) \tag{3.24}$$

Figure 3.9 shows both the upper- and the lower-bound arrival curve. Compared to periodic event streams (see Figure 3.5), the uncertainty has increased.

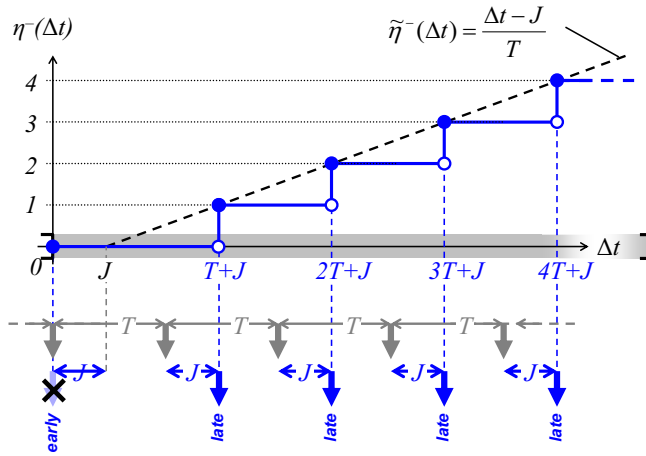



*Figure 3.8.* Lower Bound Arrival Curve of Periodic Events with Jitter

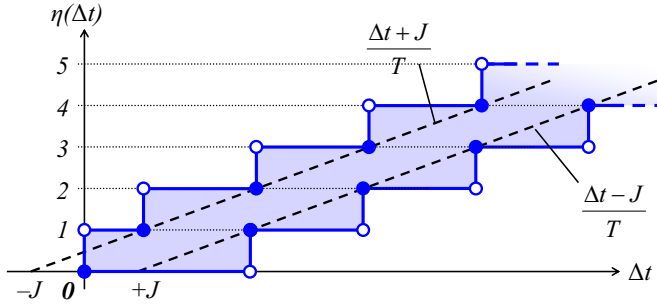



*Figure 3.9.* Both Arrival Curves of Periodic Events with Jitter

### 3.5.2 The $\delta(n)$ Functions

In contrast to strictly periodic events where the distance between any two events was constant, events with jitter can be "early" or "late". Subsequently, the distance between events can differ.

**Minimum Distance.** The minimum distance $\delta^-(2)$ between 2 events in a periodic stream with jitter intuitively is the period minus the jitter. The difference between 3 events is one period larger, and so on. This information was

already used to construct the scenario with the maximum number of events, shown in Figure 3.7.

$$\forall\, n \in \mathbb{N} \setminus \{0,1\} : \delta^-_{\mathsf{P+J}}(n) = (n-1)T - J \tag{3.25}$$

Audsley exploited this property to formulate appropriate deadlines for tasks when recurrence is prohibited. The analysis in Equations 3.19 and 3.20 illustrate the influence of the $\delta^-$ function on task deadlines:

$$r_i \leq T_i - J_i = \delta^-_{\mathsf{P+J}}(2) \tag{3.26}$$

The windowing techniques (see Equation 3.14) can also work with periodic events with jitter.

**Maximum Distance.** The maximum distance between two events is the period *plus* the jitter. The difference between three events is one period larger, and so on, leading to Equation 3.27:

$$\forall\, n \in \mathbb{N} \setminus \{0,1\} : \delta^+_{\mathsf{P+J}}(n) = (n-1)T + J \tag{3.27}$$

### 3.5.3 Event Model Definition

From the properties of periodic event streams with jitter, we can derive the formal definition of the model of periodic events with jitter:

$$\begin{aligned}
\mathcal{EM}_{\mathsf{P+J}} = \{\, \mathcal{S}_{\mathsf{P+J}}(\pi_{\mathsf{P+J}}) \quad | \quad & \pi_{\mathsf{P+J}} \in \Pi_{\mathsf{P+J}} = \\
& \{(T,J) \mid T \in \mathbb{R}^+ \setminus \{0\}, J \in \mathbb{R}^+, J < T\}\,, \\
& \eta^+_{\mathsf{P+J}}(\Delta t) = \begin{cases} \left\lceil \frac{\Delta t + J}{T} \right\rceil & \Delta t > 0 \\ = 0 & \Delta t = 0 \end{cases}, \\
& \eta^-_{\mathsf{P+J}}(\Delta t) = \max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right), \\
& \delta^-_{\mathsf{P+J}}(n) = (n-1)T - J, \\
& \delta^+_{\mathsf{P+J}}(n) = (n-1)T + J\}
\end{aligned} \tag{3.28}$$

The model of periodic events with jitter is very powerful. In addition to static, strictly periodic events, the concept of jitter allows the modeling of dynamic event streams with uncertain, only partially known behavior within a generally periodic stream. Basically, the regular nature of periodic events still lays the foundation of all predictability issues, whether with jitter or not.

Besides periodic events, there is the class of *aperiodic events* [108] which arrive irregularly. In general, such aperiodic events cannot be subject to any real-time analysis since key characteristics like the maximum number of event

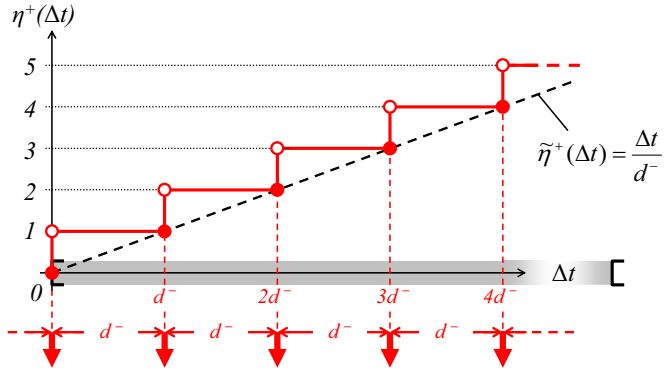



*Figure 3.10.* Upper Bound Arrival Curve of Sporadic Events

arrivals $\eta(\Delta t)$ can not be safely bounded. However, there is one sub-class of aperiodic events, called *sporadic* events, with special properties that allow a minimum of predictability and analyzability.

## 3.6 Sporadic Events

In contrast to periodic events, sporadic events arrive irregularly. There is no average period, repetition time, or frequency. However, sporadic events still allow the maximum event arrival to be bounded, since they have a known minimum separation in time between any successive events, called the *inter-arrival time* $d^-$ [108, 3].

### 3.6.1 The $\eta(\Delta t)$ Functions

**Upper-Bound Arrival Function.** The maximum number of events can be easily calculated from the inter-arrival time. Intuitively, the number is equal to the maximum number of events of a periodic stream with a period equal to the inter-arrival time. In other words, the worst-case behavior (maximum number of events) of a sporadic stream is a strictly periodic stream [108]. So, the same considerations, equations, and curves can be re-used from Section 3.3. The curve is shown in Figure 3.10, and the $\eta^+$ function is given by Equation 3.29.

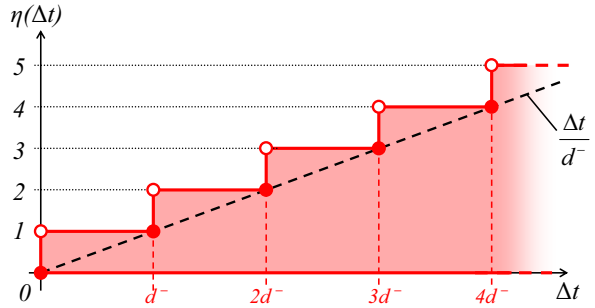



*Figure 3.11.* Both Arrival Curves of Sporadic Events

$$\begin{aligned} \forall\, \Delta t > 0 : \tilde{\eta}^{+}_{\mathsf{S}}(\Delta t) &= \frac{\Delta t}{d^{-}} \\ \forall\, \Delta t > 0 : \eta^{+}_{\mathsf{S}}(\Delta t) &= \left\lceil \tilde{\eta}^{+}_{\mathsf{S}}(\Delta t) \right\rceil = \left\lceil \frac{\Delta t}{d^{-}} \right\rceil \end{aligned} \tag{3.29}$$

**Lower-Bound Arrival Function.** Sporadic events only have a minimum distance defined, required to bound the maximum number of events. The minimum number is not bounded and must be assumed to be zero (0). That means, in the best case, there is not a single event occurrence:

$$\forall\, \Delta t > 0 : \eta^{-}_{\mathsf{S}}(\Delta t) = 0 \tag{3.30}$$

The corresponding curve is trivially zero. Both curves are combined into Figure 3.11. The figure illustrates the effect of sporadic event sources. The uncertainty linearly increases with increasing $\Delta t$, while it is constant for periodic models.

### 3.6.2 The $\delta(n)$ Functions

**Minimum Distance.** We have already mentioned that a sporadic stream shows strictly periodic properties in the worst case with maximum event arrival, given that $T_{\mathsf{P}} = d^{-}_{\mathsf{S}}$. The $\eta^{+}(\Delta t)$ functions of both streams equal for all $\Delta t$. Quite similarly, also the $\delta^{-}(n)$ function are identical. The actual function is given by Equation 3.31. This is used by the work in [121] to find termination conditions for the windowing technique.

$$\forall\, n \in \mathbb{N} \setminus \{0, 1\} : \delta^{-}_{\mathsf{S}}(n) = (n-1)d^{-} \tag{3.31}$$

**Maximum Distance.** Due to the sporadic nature, the maximum distance is *infinity*. In other words, sporadic events need not arrive at all:

$$\forall\, n \in \mathbb{N} \setminus \{0,1\} : \delta^+_{\mathsf{S}}(n) = \infty \tag{3.32}$$

### 3.6.3 Event Model Definition

The model of sporadic events is given by:

$$\begin{aligned}\mathcal{EM}_{\mathsf{S}} = \{\, \mathcal{S}_{\mathsf{S}}(\pi_{\mathsf{S}}) \quad | \quad & \pi_{\mathsf{S}} \in \Pi_{\mathsf{S}} = \{(d^-) \mid d^- \in \mathbb{R}^+ \setminus \{0\}\,, \\ & \eta^+_{\mathsf{S}}(\Delta t) = \left\lceil \frac{\Delta t}{d^-} \right\rceil, \\ & \eta^-_{\mathsf{S}}(\Delta t) = 0\,, \\ & \delta^-_{\mathsf{S}}(n) = (n-1)d^-, \\ & \delta^+_{\mathsf{S}}(n) = \infty\}\end{aligned} \tag{3.33}$$

## 3.7 Sporadically Periodic Events

Sporadically periodic events [4] represent a specialized class of sporadic events. Other work [121] uses the same model but refers to such streams as *sporadically bursty* (B), where each appearance of several events is considered a *burst*.

A stream is characterized by three parameters. An *outer period* $T^O$, defining the minimum temporal separation of the successive bursts, a *burst length* $b$ determining the maximum number of events within one burst, and the *inner period* $T^I$ defining a minimum distance between two events within a burst.

### 3.7.1 The $\eta(\Delta t)$ Functions

**Upper-Bound Arrival Function.** Again, some simple considerations help to construct a scenario with maximum number of events. The measurement interval is aligned to the first event within a burst. Subsequent events in that burst arrive as soon as possible –similar to the "early" events in periodic streams with jitter–, i. e. with their inner period. Each subsequent burst has the same properties. And the bursts re-arrive as soon/often as possible, i. e. with their outer period. A scenario with a burst length of $b = 4$ and the corresponding upper-bound arrival curve is shown in Figure 3.12. The curve illustrates bursts and idle periods. It shows the (worst-case) periodic nature "within" a burst as well as the sporadic nature of the appearance of the "bursts" itself. The $\eta^+(\Delta t)$ function is given by Equation 3.34. The equation separates: a) a number of $\lfloor \frac{\Delta t}{T^O} \rfloor$ full bursts of $b$ events each, and b) the remaining sporadic events in the remaining time interval $\Delta t - \lfloor \frac{\Delta t}{T^O} \rfloor T^O$, eventually bounded by

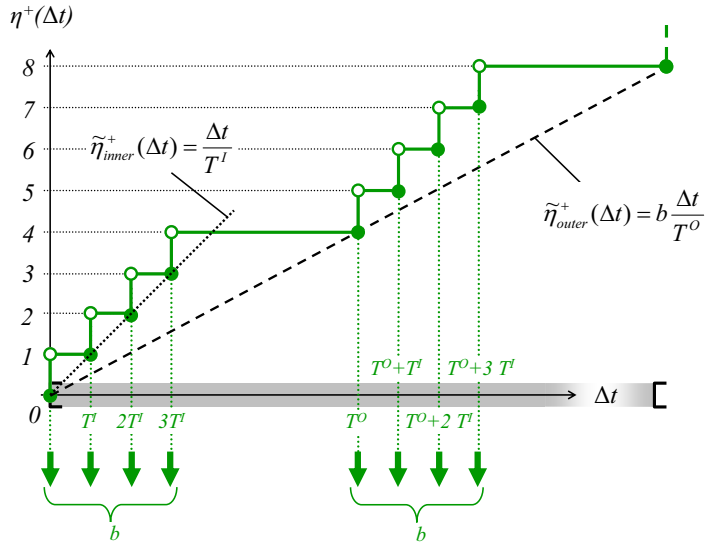



*Figure 3.12.* Upper Bound Arrival Curve of Sporadically Periodic Events

the maximum burst length $b$.

$$\forall\ \Delta t > 0 : \eta^+_{\mathsf{B}}(\Delta t) = \underbrace{\left\lfloor \frac{\Delta t}{T^O} \right\rfloor b}_{\text{full bursts}} + \underbrace{\min\left( \left\lceil \frac{\Delta t - \left\lfloor \frac{\Delta t}{T^O} \right\rfloor T^O}{T^I} \right\rceil, b \right)}_{\text{remaining events}} \tag{3.34}$$

This function can be found in the response time approaches of Tindell [121] and Audsley [4].

In contrast to the three other models, no equally simple continuous $\tilde{\eta}^+$ function exists. We can, however, provide two continuous ones, each one focusing on a different issue of sporadically periodic events. When we apply the inner period to $\tilde{\eta}^+_{\mathsf{P}}$ of equation 3.9, we obtain $\tilde{\eta}^+_{inner}$ bounding the maximum frequency during bursts in Figure 3.12. Similarly, we can find $\tilde{\eta}^+_{outer}$ when using the outer period and the burst length, providing a long-term average number of events. Both these curves are shown in Figure 3.12.

**Lower-Bound Arrival Function.** Sporadically periodic events –as a generalized class of sporadic events– do not need to appear at all. Hence, the $\eta^-$ function is assumed zero:

$$\forall\ \Delta t > 0 : \eta^-_{\mathsf{B}}(\Delta t) = 0 \tag{3.35}$$

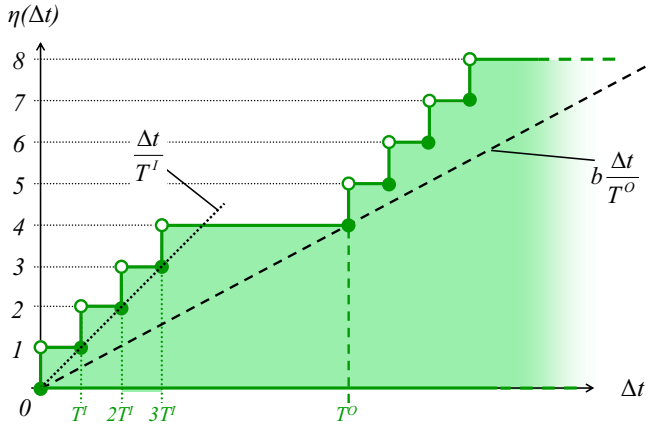



*Figure 3.13.* Both Arrival Curves of Sporadically Periodic Events

Both event arrival curves are combined in Figure 3.13.

### 3.7.2 The $\delta(n)$ Functions

**Minimum Distance.** Similar to Equation 3.34, the actual calculation of the minimum distance $\delta^-(n)$ distinguishes full bursts and a remainder:

$$\forall\, n \in \mathbb{N} \setminus \{0,1\} : \delta^-_{\mathsf{B}}(n) = \underbrace{\left\lfloor \frac{n-1}{b} \right\rfloor T^O}_{\text{full bursts}} + \underbrace{((n-1) - \left\lfloor \frac{n-1}{b} \right\rfloor b) T^I}_{\text{remaining events}} \tag{3.36}$$

**Maximum Distance.** The maximum event distance of any sporadic event stream is infinity:

$$\forall\, n \in \mathbb{N} \setminus \{0,1\} : \delta^+_{\mathsf{B}}(n) = \infty \tag{3.37}$$

### 3.7.3 Event Model Definition

The model of sporadically periodic events is given by:

$$
\begin{aligned}
\mathcal{EM}_{\mathsf{B}} = \{\, & \mathcal{S}_{\mathsf{B}}(\pi_{\mathsf{B}}) \mid \\
& \pi_{\mathsf{B}} \in \Pi_{\mathsf{B}} = \{(T^O, T^I, b) \mid T^O \in \mathbb{R}^+ \setminus \{0\}, T^I \in \mathbb{R}^+, \\
& \qquad b \in \mathbb{N}^+ \setminus \{0\}, T^O \geq T^I \times b\}\,, \\
& \eta^+_{\mathsf{B}}(\Delta t) = \left\lfloor \frac{\Delta t}{T^O} \right\rfloor b + \min\left( \left\lceil \frac{\Delta t - \left\lfloor \frac{\Delta t}{T^O} \right\rfloor T^O}{T^I} \right\rceil, b \right), \\
& \eta^-_{\mathsf{B}}(\Delta t) = 0\,, \\
& \delta^-_{\mathsf{B}}(n) = \left\lfloor \frac{n-1}{b} \right\rfloor T^O + ((n-1) - \left\lfloor \frac{n-1}{b} \right\rfloor b)\, T^I, \\
& \delta^+_{\mathsf{B}}(n) = \infty\}
\end{aligned}
\tag{3.38}
$$

## 3.8 Summary

The use of abstract event stream is essential for many scheduling analysis techniques such as RMS (rate-monotonic scheduling). We have defined event streams and event models, and we have reviewed the four most popular event models in literature: *strictly periodic* (P), *periodic with jitter* (P+J), *sporadic* (S), and *sporadically periodic* or *sporadic bursts* (B). These models require only a few key parameters in order to describe the behavior of a stream.

Using examples such as in Equations 3.7 and 3.22, we have shown *how* these parameters are used in scheduling analysis. Four characteristic functions apply to all event streams. For instance, the maximum number of events $\eta^+(\Delta t)$ is required to determine the number of preemptions of a task. The minimum distance between events $\delta^-(n)$ is used to restrict task deadlines (see Equation3.26) or to correctly capture for task recurrence as in Equation 3.14. These two functions capture the key event stream properties in the area of classical scheduling analysis that targets maximum (worst-case) task and communication response times. Approaches that target distributed systems additionally require the minimum number of events and the maximum distance, to determine minimum response times, in turn required to resolve system-level scheduling anomalies. We have explained these effects in Section 2.2.2. We conclude this chapter with an overview of the characteristic functions of the four event models in Table 3.1.

| | $\pi$ | $\eta^+(\Delta t)$ | $\eta^-(\Delta t)$ | $\delta^-(n)$ | $\delta^+(n)$ |
|---|---|---|---|---|---|
| P | $T$ | $\left\lceil \frac{\Delta t}{T} \right\rceil$ | $\left\lfloor \frac{\Delta t}{T} \right\rfloor$ | $(n-1)T$ | $(n-1)T$ |
| P+J | $T, J$ | $\left\lceil \frac{\Delta t + J}{T} \right\rceil$ | $\max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right)$ | $(n-1)T - J$ | $(n-1)T + J$ |
| S | $d^-$ | $\left\lceil \frac{\Delta t}{d^-} \right\rceil$ | 0 | $(n-1)d^-$ | $\infty$ |
| B | $T^O, T^I, d$ | $\left\lfloor \frac{\Delta t}{T^O} \right\rfloor b+$ $\min\left(\left\lceil \frac{\Delta t - \lfloor \frac{\Delta t}{T^O} \rfloor T^O}{T^I} \right\rceil, b\right)$ | 0 | $\left\lfloor \frac{n-1}{b} \right\rfloor T^O+$ $((n-1)-$ $\left\lfloor \frac{n-1}{b} \right\rfloor b)T^I$ | $\infty$ |

*Table 3.1.* Parameters and Characteristic Functions of the Four Most Popular Event Models

Chapter 4

# OUTPUT EVENT MODELS

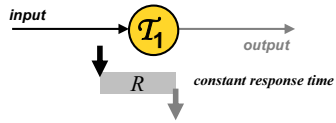
We have seen how event models are used to capture task activation. Strictly periodic events, for instance, might be generated by a timer interrupt. The other models, however, aim at capturing the partially non-deterministic behavior of *event sources*, i. e. other tasks that *produce* the events that activate tasks. This already shows that event timing needs not only be considered at task input but also at task's outputs.

This chapter investigates task input-output timing behavior and specifically analyzes the output event generation. This is absolutely necessary for the component integration step mentioned in Section 2.5. One task's output becomes another task's input. We also need appropriate *output event models*, otherwise two models cannot be reasonably coupled [101, 98]. The key goal in this context is a consistent and systematic use of event models.

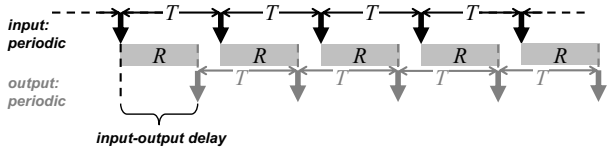
We use expressive examples to introduce the basic concepts of output modeling. We start with simple, constrained task configurations such as periodic tasks with deterministic timing behavior. We then gradually add more concepts, especially all possible sources of non-determinism like jitters, sporadic event sources, and conditional task execution.

## 4.1 Periodic Task Activation

We have mentioned in Section 3.1 that task activation can always be modeled as event triggered, if we can properly describe the *event sources that produce these events*. However so far, only relatively simple event sources have been considered, such as a system timer with a fixed frequency. Such event generation is a form of output production, the timer outputs events, and can be described using the *periodic event model*. In the local analysis techniques mentioned in Section 2.1, such event models are directly used as input event models, i. e. activating events, for a task.




(a) Constant Response Time




(b) Periodic Input and Periodic Output

*Figure 4.1.* Periodic Task with Constant Response Time

### 4.1.1 Constant Response Times

Periodic output production generally applies to all tasks that are activated periodically and have a constant input-output delay, i. e. a constant response time. In this thesis we assume that a task produces its output on completion. So the *output generation times of the $i$th output are equal to the task completion times of the $i$th task execution*:

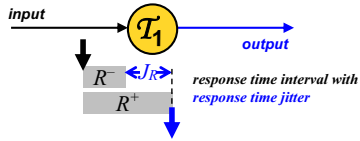
$$t_{\text{out}}(i) = t_{\text{in}}(i) + R(i) \tag{4.1}$$

A periodic task $\mathcal{T}_1$ with a constant response time together with its periodic input and output streams is shown in Figure 4.1(a). It is quite clear that, except in the case of a constant phase delay which is not considered by event models, both input and output streams have equivalent properties, they are identical:

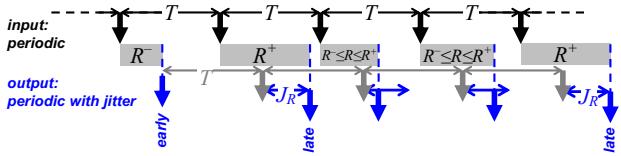
$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{1,\text{in}} = \mathcal{S}_{\mathsf{P}}(T_{\text{in}}) \tag{4.2}$$

### 4.1.2 Response Time Intervals and Output Jitter

Such a simple and static input-output behavior is, however, not commonly the case. Sophisticated architectures include pipelines and caches might stall the task, i. e. enforce wait cycles, in a seemingly non-deterministic manner. Secondly, the task might have a data-dependent control flow using `if-then-else` statements or data-dependent loop counts. Scheduling introduces delays that far exceed other architecture influences. Depending on the behavior of other tasks on the same resource, the number of preemptions can be




(a) Non-Constant Response Time




(b) Periodic Output with Jitter

*Figure 4.2.* Inheritance of Response Time Jitter at Task Output

smaller or larger. We have already seen such tasks in the introduction (see Figure 1.3). The result is a non-constant task response time, represented by an interval constrained by upper and lower response time bounds, as indicated in Figure 4.2(a):

$$R_1(i) \in R_1 = [R_1^-; R_1^+] \tag{4.3}$$

Even if activated strictly periodically, the output of such a task *inherits* the response time jitter $J_{1,R}$. Figure 4.2(b) shows an execution scenario with varying response times. The minimum or *shortest response time results in an early output* while the *maximum response leads to a late output*. So, the output jitter equals the response time jitter which is the difference between the two response time interval bounds:

$$J_{1,\text{out}} = J_{1,R} = R_1^+ - R_1^- \tag{4.4}$$

The output stream of task $\mathcal{T}_1$ is:

$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{\text{P+J}}(T_{1,\text{in}}, R_1^+ - R_1^-) \tag{4.5}$$

### 4.1.3 Inheritance of Input Jitter

Such periodic streams with jitter can activate other tasks. Figure 4.1.3 shows a chain of tasks. Task $\mathcal{T}_1$ is taken from the previous example and is known to have a strictly periodic activation and an output jitter. This output jitter turns into the input jitter of task $\mathcal{T}_2$.

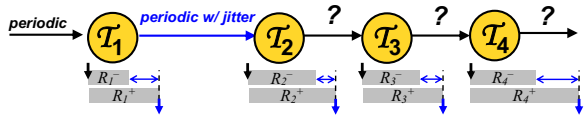



*Figure 4.3.* Non-Constant Response Times in a Task Chain

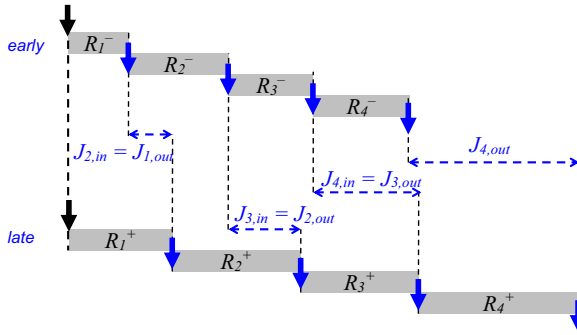



*Figure 4.4.* Early and Late Scheduling Diagrams

Tindell and Clark [122] studied such systems and they recognized that a possible input jitter is also inherited at the task's output, just as the (internal) response time jitter is inherited. In other words, the response time jitters accumulate along a task chain, where each task adds its response time jitter from input to output:

$$\begin{aligned} J_{i,\text{out}} &= J_{i,\text{in}} + J_{i,R} \\ &= J_{i,\text{in}} + (R_i^+ - R_i^-) \end{aligned} \tag{4.6}$$

Hence, the output stream is given by:

$$\mathcal{S}_{i,\text{out}} = \mathcal{S}_{\text{P+J}}(T_{i,\text{in}}, J_{i,\text{in}} + J_{i,R}) \tag{4.7}$$

The early and late execution sequence diagrams and the jitter accumulation are shown in Figure 4.3.

The jitter plays an important role in system-level scheduling. It captures the scheduling influences of one task at the input of another. This is also the basic underlying idea of the holistic analysis techniques published by Tindell and Clark [122] and Gutierrez, Palencia, and Harbour[41]. They establish response time equations for each task in the system. For dependent tasks, these equations are mutually dependent, resulting in a complex equation set to be solved.

Best-case response times $R^-$ were initially simply considered zero, resulting in unnecessarily large jitters. Since then, many improvements of best-case analysis have been proposed. Tindell [120] captured static phase information, the phase delay between the activation of two successive tasks. He called this phase *offset* [120] within a *transaction*. Palencia and Harbour [82] extended this approach to dynamic offsets. Other groups have been studying best-case analysis, too [62, 47, 92, 49]. Interestingly, most of the work is done no earlier than 30 years after Graham forecasted an increase in the importance of best-case analysis when resolving scheduling anomalies [37].

### 4.1.4 Event Streams with Large Jitters

Improving best-case analysis helps to increase the accuracy of the jitter calculation. The improvements, however, do not reduce the actual jitter, which monotonically increases along task chains and can even exceed the original period. We mentioned in Section 3.5 that many of the known scheduling and analysis techniques require jitters less than periods to be applicable. A jitter equal to the period would result in the simultaneous arrival of two task activations, in turn leading to task recurrence. Not all analysis approaches consider recurrence but require that one execution is finished before the next activation arrives.

There are few extensions. Lehoczky [69] and Tindell [121] introduced the notion of *arbitrary deadlines* and presented response time analysis techniques that are not restricted to any deadlines. Their techniques allow for large jitters, and we extend the jitter model from Section 3.5 by allowing *arbitrary jitters*:

$$\pi_{\mathsf{P+J}} \in \Pi_{\mathsf{P+J}} = \{(T, J) | T \in \mathbb{R}^+ \setminus \{0\}, J \in \mathbb{R}^+\} \tag{4.8}$$

The consequences on the characteristic functions are explained in the following sections.

#### 4.1.4.1 The $\eta(\Delta t)$ Functions

**Upper-Bound Arrival Function.** To construct the upper bound arrival scenario, we apply the well known technique from Section 3.5. We assume the first event arriving as late as possible and all others arriving as early as possible. This has already been used in constructing worst-case event arrival in the previous chapter. Figure 4.5 shows an upper bound arrival for an event stream where the jitter is larger than the period. The curve has almost the same properties as the curves of streams with jitters less than periods. So, the $\eta^+(\Delta t)$ function can be reused from Equation 3.18 in Section 3.5:

$$\forall\, \Delta t > 0 : \eta^+_{\mathsf{P+J}}(\Delta t) = \left\lceil \frac{\Delta t + J}{T} \right\rceil \tag{4.9}$$

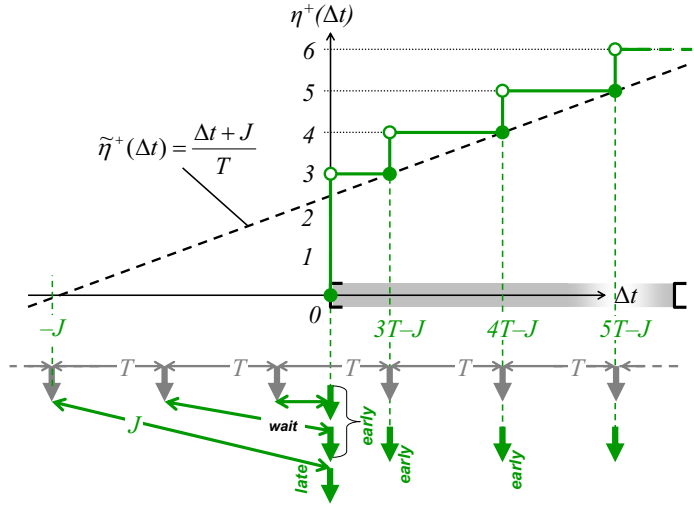



*Figure 4.5.* Upper-Bound Arrival Curve of Periodic Events with "Large" Jitter

There are, however, some important differences between jitter less than periods, and jitter equal to or larger than periods. In Figure 4.5, we see that the first three events now arrive seemingly simultaneously, i. e. within an interval of $\Delta t = 0$ time units. But for reasons explained in Section 3.3.1, the $\eta(\Delta t)$ function is defined to be zero at $\Delta t = 0$. To calculate the number of events that arrive simultaneously ($n_0$), we need to determine the limes for $\Delta t \to 0, \Delta t > 0$:

$$n_0 = \lim_{\Delta t \to 0, \Delta t > 0} \eta^+(\Delta t) = \left\lfloor \frac{J}{T} \right\rfloor + 1$$

Another important observation in Figure 4.5 is that the second and third event also experience a delay, although they should be assumed early. This is because events in a stream must usually maintain the order in which they arrive, i. e. events are not allowed to "overtake" each other and thus have to "wait". This preserves the causality of input and output streams of a task, and it is also the best strategy to reduce the maximum "waiting time" of each individual event. Hence, the second and third event cannot be early in the sense of immediately. This is an important property of large jitters. In other words, there is an implicit minimum inter-arrival time of zero that preserves the order of events.

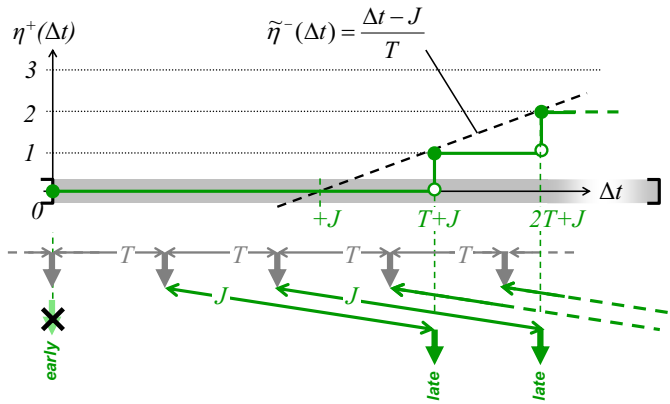



*Figure 4.6.* Lower- Bound Arrival Curve of Periodic Events with "Large" Jitter

**Lower-Bound Arrival Function.** The lower bound arrival curve is shown in Figure 4.6. Clearly, the function known from small jitters (Sec. 3.5) applies which was already given by Equation 3.24:

$$\forall\, \Delta t > 0 : \eta^-_{\mathsf{P+J}}(\Delta t) = \max\left(0, \left\lceil \frac{\Delta t - J}{T} \right\rceil\right) \tag{4.10}$$

#### 4.1.4.2 The $\delta(n)$ Functions

The fact that several events can arrive simultaneously also affects the $\delta^-(n)$ function. Other than in Equation 3.25, the minimum distance between two events can be zero. In the above example it is even zero for $n = 3$. The actual $\delta^-(n)$ function is given by Equation 4.11:

$$\forall\, n \in N, n \geq 2 : \delta^-_{\mathsf{P+J}}(n) = \max\left(0, (n-1)T - J\right) \tag{4.11}$$

The maximum distance between events is not affected in concept. The equations from the jitter model as introduced in Section 3.5 are still valid:

$$\forall\, n \in N, n \geq 2 : \delta^+_{\mathsf{P+J}}(n) = (n-1)T + J \tag{4.12}$$

## 4.2 Implications of Large Jitters

A key property of streams with large jitters (we will use the formulation *large jitter* for jitter larger than the period) is the possibly simultaneous arrival of events. However, tasks are not usually activated simultaneously but with a certain delay. The execution sequence of Figure 4.7 provides an example.

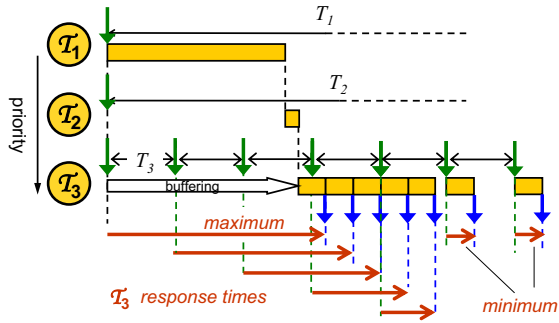



*Figure 4.7.* An Example of Output Events with "Large" Jitter

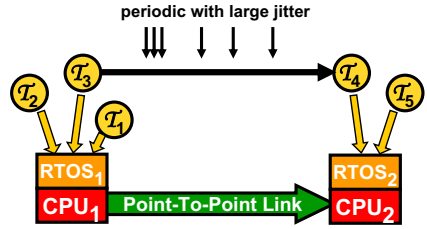



*Figure 4.8.* Distributed System Example

It contains a zoomed view of Figure 1.4 and shows the worst-case buffering scenario of task $\mathcal{T}_3$.

All tasks are activated periodically without any input jitter. Scheduling follows static priorities, $\mathcal{T}_1$ has the highest and $\mathcal{T}_3$ the lowest priority. The periods are $T_1$, $T_2$, and $T_3$, respectively. Task $\mathcal{T}_3$ is blocked for several periods and then executes several times in a row (bursty). The scheduling diagram illustrates that the response time jitter (the best-case to worst-case difference) far exceeds the period. In other words, task $\mathcal{T}_3$ has a large output jitter. We will now analyze the effect of this large output jitter on the scheduling and analysis of other tasks.

### 4.2.1 Propagation of Large Jitters

Figure 4.8 shows a distributed system. $\mathsf{CPU}_1$ scheduling has just been analyzed above. We use the output stream of $\mathcal{T}_3$ to determine the activation and

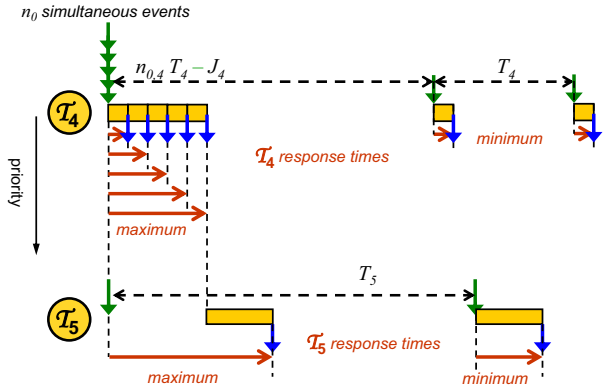



*Figure 4.9.* Scheduling with Simultaneous Task Activation due to Large Jitters

execution behavior of $\mathcal{T}_4$ in $\mathsf{CPU}_2$ scheduling analysis. First, we must propagate the stream parameters:

$$\begin{aligned} \mathcal{S}_{4,\text{in}} &= \mathcal{S}_{3,\text{out}} \\ &= \mathcal{S}_{\mathsf{P+J}}(T_3, J_{3,R}) \end{aligned}$$

Now, we can perform all kinds of scheduling analyses based on this $\mathcal{T}_4$ input stream representation. We assume that scheduling follows static priorities where task $\mathcal{T}_4$ has a higher priority than task $\mathcal{T}_5$. Figure 4.9 shows a scheduling diagram of the second resource.

Due to the large jitter, worst-case scheduling analysis for both tasks must assume the simultaneous arrival of $n_{0,4}$ task $\mathcal{T}_4$ activations, just as introduced in Section 4.1.4.1. In effect, task $\mathcal{T}_4$ is simultaneously activated and the corresponding executions interfere with each other. This has two important consequences. Firstly, task $\mathcal{T}_4$ has a large maximum response time and –again– a large output jitter. Secondly, this burst execution also leads to a large maximum response time for task $\mathcal{T}_5$. After the burst phase is over, the first task activation under "normal" conditions arrives at $n_{0,4}\ T_4 - J_4$ after the beginning of the burst.

It is quite obvious that the simultaneous arrival of multiple task activations has a major (negative) impact on the maximum response times and subsequently to the response time jitters and the overall jitters in the system; and large jitters, in turn, lead to the assumption of simultaneous events. The examples show that scheduling dependencies and input jitter inheritance and accu-

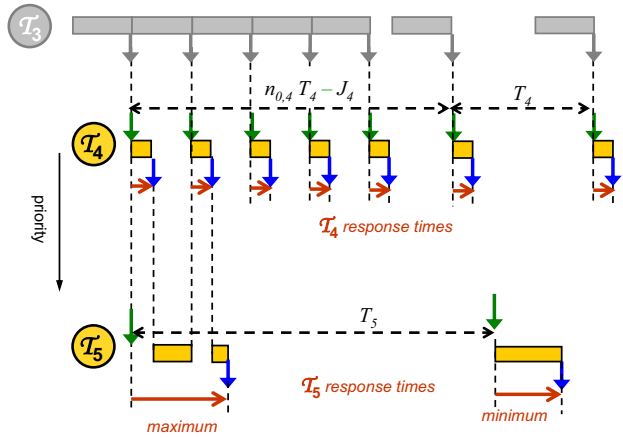



*Figure 4.10.* Scheduling with Large Jitter but Bounded Inter-Arrival Time

mulation (see Sec. 4.1.3) can quickly lead to extremely large jitters and subsequently to high transient loads and wide response time intervals. *Reducing the pessimism in the jitter analysis is a critical task, since it results in less pessimistic response times. Tighter response time intervals, in turn, result in tighter jitters.*

### 4.2.2 Limitations and Inefficiencies

In practice, simultaneous activations of the same task rarely occur as a result of jitter. When we take a closer look at the execution and scheduling of the tasks, particularly task $\mathcal{T}_3$ on the resource $\mathsf{CPU}_1$ (Fig. 4.7) , we see that the output events are not produced simultaneously, although it executes several times in a row. Hence, $\mathcal{T}_4$ on resource $\mathsf{CPU}_2$ is not activated simultaneously. Figure 4.10 shows a scheduling scenario where activations arrive one after another instead of simultaneously. For comprehensibility, the scheduling of task $\mathcal{T}_3$ on the $\mathsf{CPU}_1$ is included in the figure. Task $\mathcal{T}_4$ is activated when task $\mathcal{T}_3$ finishes.

Compared to the situation of simultaneous activation, we observe vast improvements in worst-case response times for both tasks, $\mathcal{T}_4$ and $\mathcal{T}_5$. Task $\mathcal{T}_4$ does not interfere with itself and the variations in the response time are little, if we assume a relatively constant core execution time. Furthermore, the same effect (no burst execution of $\mathcal{T}_4$) allows task $\mathcal{T}_5$ to exploit the "gaps" in the schedule, so $\mathcal{T}_5$ experiences at most two preemptions by $\mathcal{T}_4$, compared to five in the case of simultaneous $\mathcal{T}_4$ activation (see Fig. 4.9). For a better under-

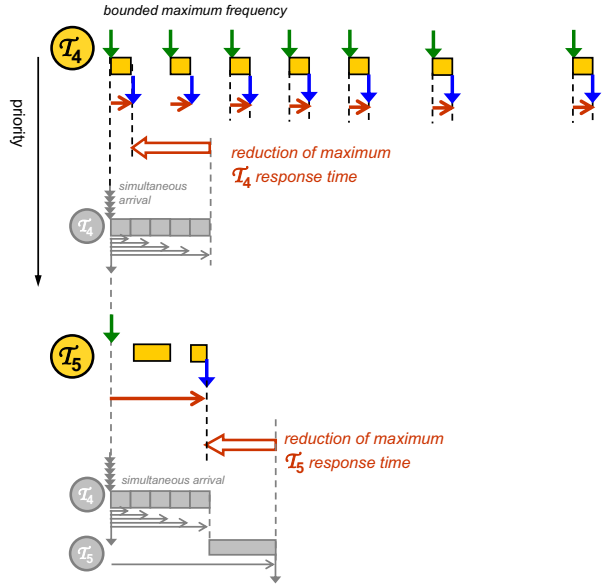



*Figure 4.11.* Comparison between two Schedules

standing, Figure 4.11 shows a detailed comparison between the two schedules. The reductions in the maximum response time bound are highlighted. These directly translate into reductions of the output jitter pessimism.

This optimization was possible since task $\mathcal{T}_3$ does not output events simultaneously, but with a certain amount of time between two successive events. The knowledge that there is a time separation between the events originates from detailed information about the execution and scheduling of $\mathcal{T}_3$ on $\mathsf{CPU}_1$. Unfortunately, this information is not preserved in the output stream representation, simply because the model of periodic events with jitter is unable to capture this property. In other words, the information is lost in the output stream representation. Subsequently, $\mathsf{CPU}_2$ analysis must assume simultaneous event arrival, with the known inefficiencies.

### 4.2.3 Modeling Alternatives

The model of periodic events with large jitter has critical limitations in its expressive power. In order to exploit the non-simultaneous arrival of events in scheduling analysis of $\mathsf{CPU}_2$, we need an event stream representation that utilizes such information.

We have summarized in Section 3.7 the model of sporadically periodic events or sporadic bursts. This event model has a notion of a *minimum inter-arrival time* or *inner period* that provides the required concepts to prevent simultaneous event arrival. The model thereby bounds the maximum *transient* event frequencies during bursts, just as observed in the above example. So, this event model is able to capture the non-simultaneous property of output events.

But the model of sporadically periodic events has another limitation: it is based on sporadic events and is thus unable to capture the periodic nature of the original input stream. In effect, best case analysis potentially assumes too few preemptions, resulting in underestimations for the minimum response time of tasks, in turn leading to larger output jitters. So, what else can we do?

In Figure 4.7, we have observed a known temporal separation between events in periodic streams with jitter. Therefore, we also introduce the concept of minimum inter-arrival to periodic models with jitter as a direct extension for large jitter support.

Quite interestingly, this extension has never been proposed before. This means that, as yet, no event model exists to capture the *effects of increasing jitter in distributed, periodic systems* to a reasonable level of accuracy. Conversely, the existing models seem more a collection of independently defined formalisms rather than a sound and systematic way to capture the key effects of scheduling and execution in distributed environments. These insufficiencies lead to overly conservative results which considerably degrades system-level analysis possibilities.

## 4.3 A New Model: Periodic Events with Burst

We introduce the idea of a minimum inter-arrival time or inner period to the model of periodic events with jitter. The result is a very powerful model which we call *periodic events with burst* (P+B) as a direct extension of the standard jitter model. A periodic stream with burst is characterized by three parameters. A period $T$, a jitter $J$, and a *minimum event distance* $d$. Simply speaking, the model captures a normal jitter, just as introduced in Section 3.5. Additionally the maximum transient event frequency is bounded, just as in the model of sporadically periodic events (see Section 3.7).

The event model of periodic events with burst is given by:

$$\begin{aligned}\mathcal{EM}_{\mathsf{P+B}} = \{\, & \mathcal{S}_{\mathsf{P+B}}(\pi_{\mathsf{P+B}}) \mid \\ & \pi_{\mathsf{P+B}} \in \Pi_{\mathsf{P+B}} = \{(T, J, d) \mid T \in \mathbb{R}^+ \setminus \{0\}, J \in \mathbb{R}^+, d \in \mathbb{R}^+\}\,, \\ & \eta^+_{\mathsf{P+B}}(\Delta t), \eta^-_{\mathsf{P+B}}(\Delta t), \delta^-_{\mathsf{P+B}}(n), \delta^+_{\mathsf{P+B}}(n)\}\end{aligned} \tag{4.13}$$

The four characteristic functions are introduced in the following sections.

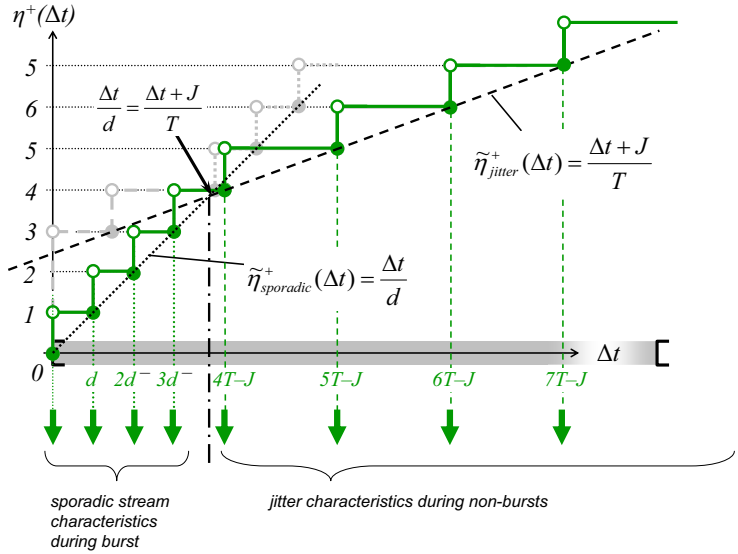



*Figure 4.12.* Upper-Bound Arrival Curve of Periodic Events with Burst: "Large" Jitter and a Minimum Distance

### 4.3.1 The $\eta(\Delta t)$ Functions

**Upper-Bound Arrival Function.** Figure 4.12 shows the resulting upper bound arrival curve. We can distinguish two regions for $\Delta t$, each region dominated by another effect. Large values of $\Delta t$ reflect the *long term behavior* of the stream that is periodic with jitter. Under this "normal" behavior, the stream is characterized by its large jitter, just as mentioned in Section 4.1.4. The curve can be described by Equation 4.9.

Small $\Delta t$s capture the behavior during bursts, and the worst-case arrival is dominated by the minimum event distance $d$ to bound the maximum transient frequency during the burst. Such behavior is already known from the two sporadic models in Sections 3.6 and 3.7, and the curve is given by Equation 3.29. After the burst has finished, the event stream returns to its "normal" behavior.

Basically, both effects (and the corresponding $\eta^+$ functions) overlap each other. Both bound the maximum number of events. The resulting $\eta^+(\Delta t)$ function is the minimum of the two individual functions:

$$\forall\, \Delta t > 0 : \eta^+_{\mathsf{P+B}}(\Delta t) = \min\left(\left\lceil \frac{\Delta t}{d} \right\rceil, \left\lceil \frac{\Delta t + J}{T} \right\rceil\right) \tag{4.14}$$

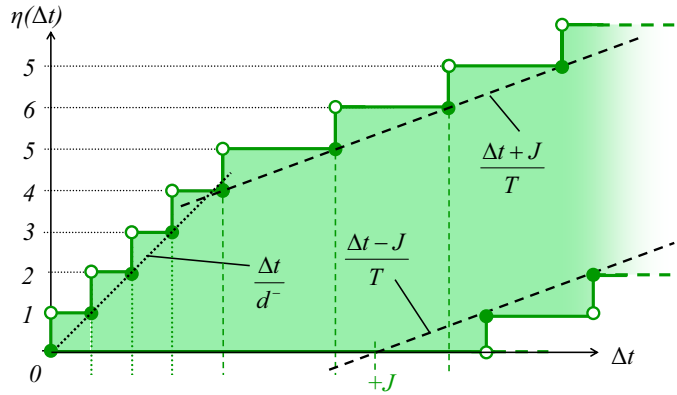



*Figure 4.13.* Upper- and Lower-Bound Arrival Curves of Periodic Events with Burst

The transition from the "burst" phase into the "normal" operation appears at the intersection of the curves of the two regions; more precisely, at the intersection of the corresponding continuous curves:

$$\begin{aligned} & \tilde{\eta}^{+}_{\text{sporadic region}}(\Delta t) &=& \tilde{\eta}^{+}_{\text{jitter region}}(\Delta t) \\ \Leftrightarrow \quad & \tfrac{\Delta t}{d} &=& \tfrac{\Delta t + J}{T} \\ \Leftrightarrow \quad & \Delta t &=& \tfrac{J\ d}{T - d} \end{aligned} \tag{4.15}$$

**Lower-Bound Arrival Function.** The minimum distance has been introduced to bound the maximum event frequencies. The lower bound function usually deals with minimum frequencies and is not influenced by the newly added property, and it equals the one given by Equation 4.10 in Section 4.1.4:

$$\forall\, \Delta t > 0 : \eta^{-}_{\mathsf{P+B}}(\Delta t) = \max\left(0, \left\lceil \frac{\Delta t - J}{T} \right\rceil\right) \tag{4.16}$$

The corresponding $\eta^{-}(\Delta t)$ curve has already been introduced in Figure 4.6. Both upper and lower bound arrival curves are shown in Figure 4.13.

### 4.3.2 The $\delta(n)$ Functions

**Minimum Distance.** Since we have introduced a parameter bounding the minimum distance between two successive events, the $\delta(n)$ functions also change compared to the function from large jitters as introduced in Section 4.1.4. Similar to the $\eta^{+}$ function, two regions can be distinguished, each dominated

by another effect. During the burst phase, the sporadic nature of minimum event distance dominates, while the period and jitter dominate the other region. The function is given by the following equation:

$$\forall\, n \in N, n \geq 2 : \delta^-_{\mathsf{P+B}}(n) = \max\left((n-1)d\ ,\ (n-1)T - J\right) \tag{4.17}$$

**Maximum Distance.** Similar to the lower-bound event arrival function, the maximum distance is not affected by the proposed extension, and the function from the "normal" jitter model as given by Equation 3.27 applies:

$$\forall\, n \in N, n \geq 2 : \delta^+_{\mathsf{P+B}}(n) = \delta^+_{\mathsf{P+J}}(n) = (n-1)T + J \tag{4.18}$$

### 4.3.3 Bounding the Minimum Output Distance

During bursts, several executions or instances of a single task interfere with each other, as seen in Figures 4.7 and 4.9, they are re-activated before previous activations have completed. This effect is known as *task recurrence* [69, 121]. In practice, such situations could eventually be handled in several ways including the termination (kill) of the current task instance or a real preemption. Such implementations, however, do not guarantee causality between input and output event streams, and they are not considered further here.

We want to guarantee input-output causality, hence a newly arriving task activation must be blocked until all preceding activations have successfully completed. Examples are given in Figures 4.7 and 4.9. A task requires at least its minimum response time to execute. Hence, the minimum response time represents an additional lower bound on the minimum output distance during *output bursts*. Finally, there is a third bound that might dominate the output during *input bursts*:

$$\begin{aligned}
\mathcal{S}_{i,\text{out}} = \quad \mathcal{S}_{\mathsf{P+B}}\Big( \quad & T_{i,\text{out}} = T_{i,\text{in}}\ , \\
& J_{i,\text{out}} = J_{i,\text{in}} + J_{i,R}\ , \\
& d_{i,\text{out}} = \max\big(\underbrace{T_{i,\text{out}} - J_{i,\text{out}}}_{\text{as with jitter}},\ \underbrace{R^-_i}_{\substack{\text{during}\\ \text{output}\\ \text{bursts}}}\ ,\ \underbrace{d_{i,\text{in}} - J_{i,R}}_{\text{during input bursts}}\ \big)\Big)
\end{aligned} \tag{4.19}$$

The third bound is related to transient frequencies that increase from input to output, similar to the increase in jitter, and is explained in detail in the next section.

## 4.4 Sporadic Task Activation

While periodic events (with or without jitter) have a certainly deterministic behavior, i. e. their arrival can be predicted to a certain extent, sporadic events

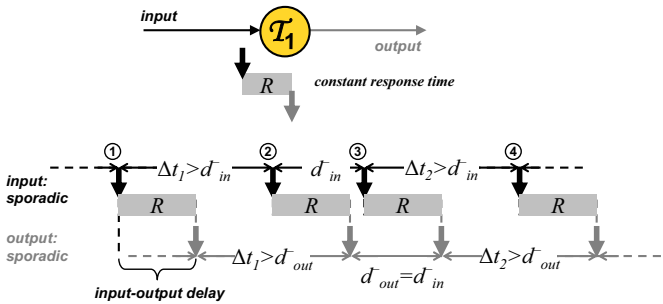



*Figure 4.14.* Sporadic Task Execution with Constant Response Time

arrive seemingly irregularly. Sporadic task activation models have proven powerful in modeling the behavior of such non-deterministic event sources, for instance customer requests to servers. Details have already been reviewed in Section 3.6. This section focuses on the effect of execution and scheduling on the properties of sporadic event streams, just as we have done above for periodic streams.

### 4.4.1 Constant Response Time

A constant response time represents a constant input-output delay, and the task output equals the task input except a constant phase delay. We have already observed this behavior for periodic events in Section 4.1.1. Hence, the output stream equals the input stream:

$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{1,\text{in}} = \mathcal{S}_{\text{S}}(d^{-}_{1,\text{in}}) \tag{4.20}$$

### 4.4.2 Response Time Interval

Sporadic events that enter a system experience the same types of distortion as periodic events. The stream timing changes from a task's input to the task's output, response time intervals lead to increasing uncertainty about the output event arrival curves. We have already analyzed the effects of non-constant response times to task input-output timing in Section 4.1.2. In the case of periodic events, we could easily use the known jitter event model to capture this effect. But the concept of a jitter is unknown for sporadic events which only define a minimum inter-arrival time. So, we need a different approach.

The worst-case situation is shown in Figure 4.15. Two effects contribute to the "worst-caseness" here: a) the two task activations arrive with their minimum inter-arrival time, i. e. no other two successive task activations will ever arrive closer in time, and b) a "late" output event is followed by an "early"

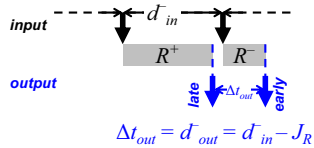



*Figure 4.15.* Worst-Case Output Timing of Sporadic Tasks

one, i. e. the response time jitter reduces the event distance from input to output further, just as already known from jitters (see Sec. 4.1.3). Essentially both mentioned effects must coincide to yield the worst case. A larger input distance will result in a larger output distance, as will a different response time distribution. From these observations, we can calculate the sought-after minimum output inter-arrival time:

$$\mathcal{S}_{i,\text{out}} = \mathcal{S}_\text{S}(d^-_{i,\text{out}} = d^-_{i,\text{in}} - J_{i,R}) \tag{4.21}$$

This also represents the third bound in Equation 4.19.

### 4.4.3 Decreasing Inter-Arrival Times

Similar to increasing jitters, the minimum inter-arrival time usually decreases from input to output, the amount of decrease is the response time jitter. If we consider a chain of several tasks, we can easily imagine that the reduction given by Equation 4.21 could theoretically lead to negative results for $d_\text{out}$. Clearly, negative event distances simply do not make any sense, and a zero minimum inter-arrival time translates into a infinite worst-case event frequency, we would need to assume an infinite number of events all arriving simultaneously.

We have already discussed the problem of simultaneous event arrival in Section 4.2.2 for the jitter model. We introduced the notion of an output event distance as an additional bound to the basic jitter model (which we now call the model of periodic events with burst). And we could show that the minimum response time bounds this output distance. This bound not only applies to the output event distance in the newly introduced model of periodic events with burst (see Equation 4.19). It is a *general bound* on the output timing of *any task* and also applies to the model of sporadic events, and Equation 4.21 turns into:

$$\mathcal{S}_{i,\text{out}} = \mathcal{S}_\text{S}\left(d^-_{i,\text{out}} = \max\left(d^-_{i,\text{in}} - J_{i,R}, R^-_i\right)\right) \tag{4.22}$$

### 4.4.4 Sporadically Periodic Task Activation

The model of sporadically periodic events (or sporadic bursts) from Section 3.7 also defines a minimum inter-arrival time, also called *inner period* $T^I$ (see Section 3.7). The minimum response time represents a bound to this inner

period, too. Furthermore, the response time jitter, introduced in Section 4.1.2, influences both the inner and the outer output period:

$$\begin{aligned}\mathcal{S}_{i,\text{out}} = \quad \mathcal{S}_{\mathsf{B}}\Big( \quad & T^O_{i,\text{out}} = \max\left(T^O_{i,\text{in}} - J_{i,R},\ \underbrace{b_{i,\text{in}} \times T^I_{i,\text{out}}}_{\text{mathematical bound}}\ \right), \\ & T^I_{i,\text{out}} = \max\left(T^I_{i,\text{in}} - J_{i,R}, R_i^-\right), \\ & b_{i,\text{out}} = b_{i,\text{in}} \Big) \end{aligned} \tag{4.23}$$

The *mathematical bound* on the outer period captures the fact that the maximum average frequency $\frac{b}{T^O}$ does not exceed the maximum transient frequency bounded by $\frac{1}{T^I}$.

## 4.5 Conditional Output Generation

The previous section focused on the propagation of sporadic event streams that are inputed into a system from its environment. However, sporadic environments are not the only source for sporadic events, they can also result from originally periodic sources in the presence of conditional communication.

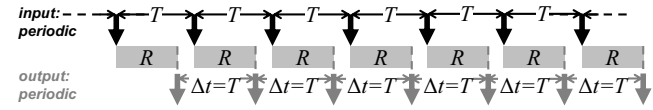
Conditional output generation means that a task does or does not produce output events, and the decision depends on the results of some calculations internal to the task, for instance within either the `then` or the `else` block of an `if-then-else` statement. In other words, the output production depends on the internal functionality of the task. Scheduling analysis often abstracts from such internal functional details and only looks at relatively simple corner cases of task execution without considering the conditions that lead to either case.

### 4.5.1 Constant Response Times

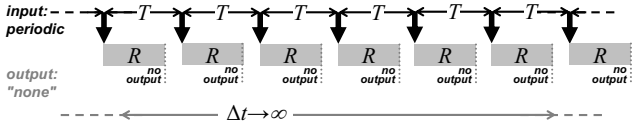
Again, we start with the situation of a constant response time. We can identify two corner-case scenarios. In the scenario of Figure 4.16(a), the task always produces an event at completion time. This obviously results in a purely periodic output, just as introduced in the case of unconditional output production (see the introduction of this chapter). The other extreme is a scenario in which the task never produces any output. This is illustrated in Figure 4.16(b).

One already introduced model that captures such behavior is the model of sporadic events. Figure 3.11 in Section 3.6 nicely illustrates the two behavioral corner-cases. So, *conditional output production transforms a periodic input stream into a sporadic output stream*:

$$\mathcal{S}_{i,\text{out}} = \mathcal{S}_{\mathsf{S}}(T_{i,\text{in}}) \tag{4.24}$$




(a) "Always" Output




(b) "Never" Output

*Figure 4.16.* Two Corner Cases of Conditional Output Production

### 4.5.2 Response Time Intervals

The situation is more complex when we consider response time intervals. Again, we can distinguish two corner-case scenarios. In the "always output" scenario, the output is periodic with jitter as introduced in Section 4.1.3. And for the "no output" scenario, we need some kind of sporadic model. However, a model capturing periodic with jitter as one bound and "no output" as the other bound is not known in literature. The same applies to conditional tasks that are activated periodically with burst. We have to resort to the model of strictly sporadic events. The minimum inter-arrival time is calculated according to the observations from increasing jitters in generally periodic streams. This is valid because the periodic stream with jitter represents the upper output bound, and the sporadic model defines an upper bound, only:

$$\mathcal{S}_{i,\text{out}} = \mathcal{S}_{\text{S}}\Big( \max\left(T_{i,\text{in}} - J_{i,R}, R_i^-\right)\Big) \tag{4.25}$$

### 4.5.3 Input with Jitter and Burst

If the task is already activated with a jitter (or even burst), then we can easily extend Equation 4.25:

$$\mathcal{S}_{i,\text{out}} = \mathcal{S}_{\text{S}}\Big( \max\left(T_{i,\text{in}} - J_{i,\text{in}} - J_{i,R}, d_{i,\text{in}} - J_{i,R}, R_i^-\right)\Big) \tag{4.26}$$

We see that the minimum inter-arrival time continually decreases along task chains, representing increasing frequencies. These increasing frequen-

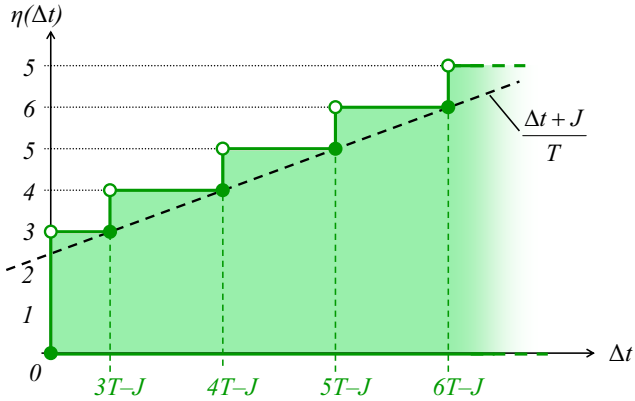



*Figure 4.17.* Upper- and Lower-Bound Arrival Curves of Sporadic Events with Jitter

cies, however, are not a property of the actual system behavior but of the event stream representation that suffers from considerable *modeling inefficiencies*.

## 4.6 New Sporadic Models with Jitter and Burst

So far, we have identified two key effects that task execution and scheduling impose on output event streams: a) jitters increase from inputs to outputs (see Sec. 4.1.3) and finally lead to bursts (Sec. 4.3), and b) conditional communication turns periodic inputs into sporadic output behavior (Sec. 4.5). The two effects are apparently orthogonal, and it is relatively straight-forward to combine the already known concepts used to capture both individual effects.

### 4.6.1 Sporadic Events with Jitter

We extend the sporadic model to also capture a possible input jitter. The model of sporadic events with jitter is defined by:

$$\begin{aligned}\mathcal{EM}_{\mathsf{S+J}} = \{\, &\mathcal{S}_{\mathsf{S+J}}(\pi_{\mathsf{S+J}}) \mid \\ &\pi_{\mathsf{S+J}} \in \Pi_{\mathsf{S+J}} = \{(T,J) | T \in \mathbb{R}^+ \setminus \{0\}, J \in \mathbb{R}^+\}\,, \\ &\eta^+_{\mathsf{S+J}}(\Delta t), \eta^-_{\mathsf{S+J}}(\Delta t), \delta^-_{\mathsf{S+J}}(n), \delta^+_{\mathsf{S+J}}(n)\}\end{aligned} \tag{4.27}$$

with the characteristic functions defined below.

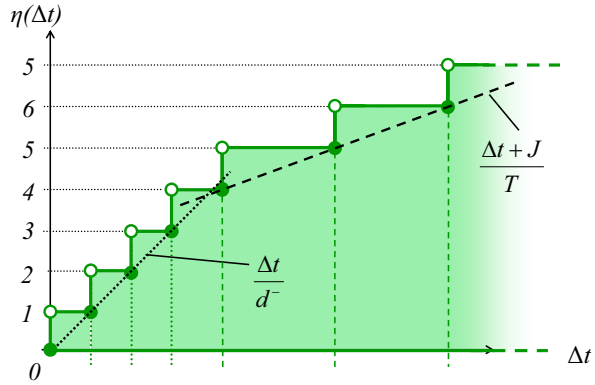



*Figure 4.18.* Upper- and Lower-Bound Arrival Curves of Sporadic Events with Burst

In the "always" case, periodic with jitter provides an upper bound on $\eta^+(\Delta t)$ and a lower bound on $\delta^-(n)$:

$$\forall\, \Delta t > 0 : \eta^+_{\mathsf{S+J}}(\Delta t) = \eta^+_{\mathsf{P+J}}(\Delta t) = \left\lceil \frac{\Delta t + J}{T} \right\rceil \tag{4.28}$$

$$\forall\, n \in N, n \geq 2 : \delta^-_{\mathsf{S+J}}(n) = \delta^-_{\mathsf{P+J}}(n) = \max\left(0, (n-1)T - J\right) \tag{4.29}$$

In the other case, the bounds are zero for $\eta^-(\Delta t)$, and infinity for $\delta^+(n)$ respectively, formally capturing the sporadic nature of the model. We call this model *sporadic events with jitter*, with the index S+J. The combined upper and lower bound arrival curves are shown in Figure 4.17.

### 4.6.2 Sporadic Events with Burst

To avoid the inefficiencies of large jitters, we also define the model of *sporadic events with burst* (S+B):

$$\begin{aligned}\mathcal{EM}_{\mathsf{S+B}} = \{\, & \mathcal{S}_{\mathsf{S+B}}(\pi_{\mathsf{S+B}}) \mid \\ & \pi_{\mathsf{S+B}} \in \Pi_{\mathsf{S+B}} = \{(T, J, d) \mid T \in \mathbb{R}^+ \setminus \{0\}, J \in \mathbb{R}^+, d \in \mathbb{R}^+\}\,, \\ & \eta^+_{\mathsf{S+B}}(\Delta t), \eta^-_{\mathsf{S+B}}(\Delta t), \delta^-_{\mathsf{S+B}}(n), \delta^+_{\mathsf{S+B}}(n)\}\end{aligned} \tag{4.30}$$

The combined upper and lower bound arrival curves are shown in Figure 4.18. The characteristic functions are given by:

$$\forall\, \Delta t > 0 : \eta^+_{\mathsf{S+B}}(\Delta t) = \eta^+_{\mathsf{P+B}}(\Delta t) = \min\left(\left\lceil \frac{\Delta t}{d} \right\rceil, \left\lceil \frac{\Delta t + J}{T} \right\rceil\right) \tag{4.31}$$

$$\forall\, \Delta t > 0 : \eta^-_{\mathsf{S+B}}(\Delta t) = 0 \tag{4.32}$$

$$\forall\, n \in N, n \geq 2 : \delta^-_{\mathsf{S+B}}(n) = \delta^-_{\mathsf{P+B}}(n) = \max\left((n-1)d\,,\ (n-1)T - J\right) \tag{4.33}$$

$$\forall\, n \in N, n \geq 2 : \delta^+_{\mathsf{S+B}}(n) = \infty \tag{4.34}$$

### 4.6.3 Sporadic Activation and Conditional Output

The combination of sporadic activation (with or without jitter or burst) and conditional output production does not lead to new output models. In one corner case, we have to assume periodic input and "always output", leading to periodic output. In the other case, we assume zero input, i. e. the task is not even executed, or it executes but does not produce output. The result is the same: zero output production. The output is in the model of sporadic events. The jitter or the burstiness can be determined just as in the case of periodic activation in Section 4.5.

## 4.7 A Six-Class Model Set

Figure 4.7 shows all output event models and indicates the transitions investigated in this chapter. We define six of these models as standard event models, that form a *six-class model set*. We derived this model set from two key observations. Jitters increase from inputs to outputs and can lead to bursts in both periodic and sporadic models. Conditional output production turns a periodic model into its sporadic "counterpart", while sporadic cannot become "more sporadic" but can induce jitter and burst.

As a key property, this six-class model set is *self-contained*, i. e. no effect brings us out of these models, and key stream information such as an average period are preserved. Just using the popular models from literature (they are highlighted in the figure) is not sufficient. They lack the expressive power to consistently handle the effects of large jitter. Burst (the consequence of large jitters) has not yet been considered in periodic models, while jitter has not yet been considered in sporadic models. The three new models *close this conceptual modeling gap* and overcome the mentioned insufficiencies. As another simplification, we can use the same parameters for sporadic and periodic models. Instead of a minimum inter-arrival time $d^-$, we refer to this value as the *sporadic period* $T$.

Two tables summarize the characteristic functions of the six models. Table 4.1 contains the upper and lower event arrival functions $\eta(\Delta t)$, while Table 4.2 summarizes the event distance functions $\delta(\Delta t)$.

The known model of "sporadically periodic" events introduced by Audsley, Tindell, and others [4, 121] does not easily fit into the other model's classification, although it has similarities with the model of sporadic events with

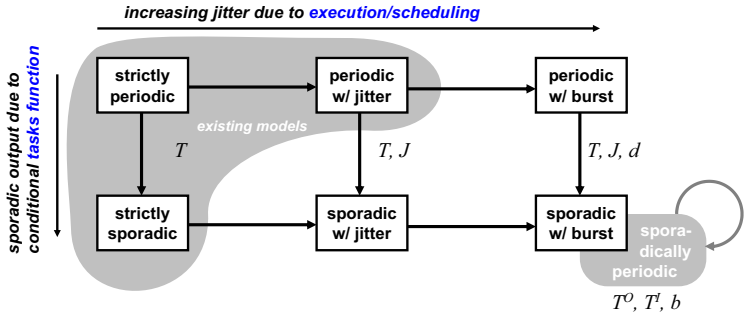



*Figure 4.19.* Self-Contained Six-Class Event Model Set

| model | $\pi$ | $\eta^+(\Delta t)$ both | $\eta^-(\Delta t)$ periodic(P) | sporadic(S) |
|---|---|---|---|---|
| P/S | $< T >$ | $\left\lceil \frac{\Delta t}{T} \right\rceil$ | $\left\lfloor \frac{\Delta t}{T} \right\rfloor$ | 0 |
| P/S+J | $< T, J >$ | $\left\lceil \frac{\Delta t + J}{T} \right\rceil$ | $\max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right)$ | 0 |
| P/S+B | $< T, J, d >$ | $\min\left(\left\lceil \frac{\Delta t + J}{T} \right\rceil, \left\lceil \frac{\Delta t}{d} \right\rceil\right)$ | $\max\left(0, \left\lfloor \frac{\Delta t - J}{T} \right\rfloor\right)$ | 0 |

*Table 4.1.* The $\eta(\Delta t)$ Functions of the New Models

| model | $\pi$ | $\delta^-(\Delta t)$ both | $\delta^+(\Delta t)$ periodic(P) | sporadic(S) |
|---|---|---|---|---|
| P/S | $< T >$ | $(n-1)T$ | $(n-1)T$ | $\infty$ |
| P/S+J | $< T, J >$ | $\max(0, (n-1)T - J)$ | $(n-1)T + J$ | $\infty$ |
| P/S+B | $< T, J, d >$ | $\max((n-1)d, (n-1)T - J)$ | $(n-1)T + J$ | $\infty$ |

*Table 4.2.* The $\delta(\Delta t)$ Functions of the New Models

burst, which we illustrate by overlapping boxes in Figure 4.7. Its interpretation of burst, however, is not based on the observation of large jitters. On the contrary, the model has no notion of jitter, and hence, it suffers from "artificially" increasing frequencies when used for modeling output streams, as we have demonstrated in Section 4.4. The model has its application in modeling complex non-deterministic environments, e. g. in control engineering or transportation systems, but is less popular for modeling internal system dependencies. However, we shall see in the next chapter, that we can find transformations between the model of sporadically periodic events into the model of sporadic events with burst, and vice versa. This allows sporadically periodic streams (and possibly others) to be captured by the newly defined six-class model.

## 4.8 Summary

In this chapter, we have systematically investigated the effects of execution, scheduling, and conditional communication on the generation of events and, subsequently, the representation of output event streams. We have seen that the existing event models have major insufficiencies in reasonably capturing the effects of large jitters, and we extended them to systematically support jitters and bursts without degrading modeling accuracy.

We defined two classes, periodic and sporadic, and we have three models in each class: strict, with jitter, and with burst. That means that we keep on with the models of periodic events with and without jitter, but define a burst as the consequence of large jitter. In order to limit the maximum transient frequency, we additionally define a minimum distance between any two events, similar to the inner period in the model of sporadically periodic events [4, 121]. The sporadic class generalizes the class of periodic models, in that one corner-case scenario is implicitly set to "no output", i. e. the $\eta^-(\Delta t)$ functions are set to zero. That means, every sporadic stream can be assumed to be the corresponding periodic stream in the worst case, which is consistent with the early application of sporadic events in real-time analysis [108].

There are three advantages of these six models over the four previously identified well known ones. First, the transition from strict to jitter to burst is intuitive. Second, periodic and sporadic streams use the same underlying models and parameters, which further reduces the complexity. The major advantage of this new set of models, however, is a conceptional one: The new set of models is self-contained with respect to input-output stream modeling. No transformation brings us out of these six models. Jitters are allowed to far exceed the period, and our new definition of burst is just the intuitive characterization of the effects of large jitters. The six-class model set appears as an efficient compromise between model simplicity and intuition on the one hand, and completeness on the other hand, and finally applicability. Because we only

applied *slight extensions* to the known models from literature, the new models can be directly –without any approximation– applied in connection with established local analysis techniques which we consider a major advantage. This sufficiency of a bounded set of intuitively structured models is likely to attract system designer's and architect's attention to the overall ideas of formal real-time analysis.

Chapter 5

# EVENT MODEL INTERFACES

In the two preceding chapters, we have introduced the models popularly found as input and output event models in scheduling analysis. Input event models capture task activation in response to incoming events. Output models capture the production of events, i. e. the production of workload for other tasks or communications. There are several popular and practically used input event models, and there are even more –and more complex– output event models. In a larger system, some components typically require certain input event models or streams, otherwise they can not be scheduled or analyzed. There are several practically important reasons for such an *input requirement*. For instance, the limited availability of analysis (tools) for a particular component forces designers to use the model dictated by the analysis (tool). Other components might have been already implemented, such as a DSP running a static periodic schedule, constraining the acceptable input further. This chapter introduces *Event Model Interfaces* that allow transformations from one event model into another to meet the mentioned input model requirements.

We will first define terms that help to capture the event model interfacing problem formally. Then, we shall derive conditions that an interface must meet, and determine compatibility tests. Finally, the existing event model interfaces are derived. We start with the six-class model set, as introduced in the previous chapter, then we consider the specialties of Audsley's and Tindell's model of sporadically periodic events [4, 121].

## 5.1 Introductory Example

Figure 5.1 shows an example of two CPUs connected through a direct point-to-point line. $\mathcal{T}_1$ on $\mathsf{CPU_1}$ sends data events to $\mathcal{T}_4$ on $\mathsf{CPU_2}$ via the event stream $\mathcal{S}_{1,4}$. Other tasks are implemented on both resources, too, but they are of no interest for this example.

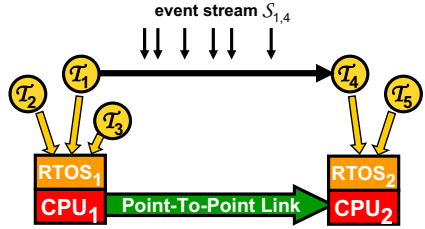



*Figure 5.1.* Introductory Example: Unidirectional Task-Task Communication via Event Streams

The operating systems (RTOS) on both resources implement different resource sharing or scheduling strategies, a simple form of heterogeneous scheduling. We make the following assumptions on both tasks:

- from the scheduling analysis of $\mathcal{T}_1$ we know that the output events are generally periodic with period $T_1$ but may experience a bounded jitter $J_1$ (e. g. resulting from preemptions by other processes). The jitter is less than the period: $J_1 < T_1$.
- $\mathcal{T}_4$ is part of an IP (intellectual property) component. No internal details are known except a maximum allowed frequency $f_4^+$, which corresponds to a sporadic input event model with a required minimum inter-arrival time (sporadic period) $T_{4,\text{req}}$.

The two event streams (periodic events with jitter versus sporadic events) are apparently *incompatible*, i. e. they are provided in different event models, and hence, have different parameters and different characteristic functions. Therefore, the output stream of task $\mathcal{T}_1$ can not be directly used for analyzing task $\mathcal{T}_4$. However, the incompatibility can be solved by *deriving the required* parameters of the *target event stream* (in this case sporadic events) *from the known* properties of the *source event stream*.

Figure 5.2 illustrates the periodic stream with jitter coming from task $\mathcal{T}_1$. It is defined by its period $T_1$ and its jitter $J_1$. The model of sporadic events defines a single parameter, the minimum period $T_4$ (minimum distance) between any two successive events. The jitter event stream at $\mathcal{T}_1$'s output is generally periodic with a period of $T_1$ but allows each event to deviate in time. Now consider two successive events, the first being as late as possible ($t_1 = t_0 + J_1$) and the second one –belonging to the next period– as early as possible, i. e. without any jitter delay ($t_2 = t_0 + T_1$). Obviously, no other two events can arrive closer in time. So, the sought-after minimum temporal separation of two successive

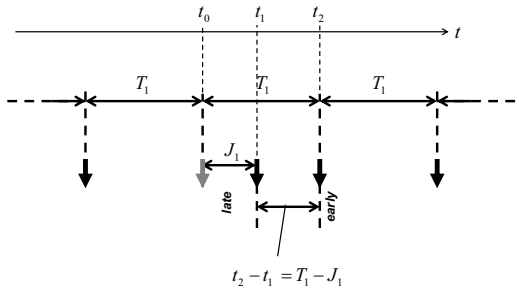



*Figure 5.2.* Worst-Case Event Timing

events with jitter is given by the following equation:

$$T_{4,\text{in}} = t_2 - t_1 = T_1 - J_1 \tag{5.1}$$

This parameter transformation represents an *event model interface*. We introduce the terms *source stream* and *target stream* to distinguish between the source and target of the transformation, and formally define event model interfaces.

**Definition 5.1.** ***(Event Model Interface)***
*An Event Model Interface $\mathcal{EMIF}$ is a function that maps a source event stream provided in a source event model, to a target event stream in the required target model.*

$$\begin{aligned} \mathcal{EMIF}_{source \rightarrow target} &: \mathcal{EM}_{source} \mapsto \mathcal{EM}_{target} \\ \mathcal{S}_{target} &= \mathcal{EMIF}_{source \rightarrow target}(\mathcal{S}_{source}) \end{aligned} \tag{5.2}$$

■

Depending on the specific properties of the involved event models, the domain of an interface, i. e. the event streams that can be successfully transformed, might be constrained to only a subset of the entire event model, i. e. not all streams in the source model can be transformed into a stream in the target model. For instance, the jitter must be less than the period in our example, otherwise Equation 5.1 would yield a zero or negative minimum period for the sporadic stream. The full definition of the event model interface of the example is:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P+J} \rightarrow \mathsf{S}} &: \{\mathcal{S}_{\mathsf{P+J}}(T, J) \in \mathcal{EM}_{\mathsf{P+J}} | J < T\} \mapsto \mathcal{EM}_{\mathsf{S}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = T_{\text{source}} - J_{\text{source}}) \end{aligned} \tag{5.3}$$

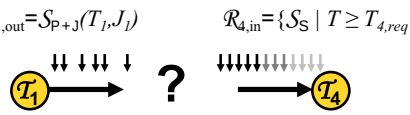



*Figure 5.3.* Problem Illustration

We have *manually* derived an appropriate parameter $T_{4,\text{in}}$ from the known parameters of task $\mathcal{T}_1$'s output. A sporadic stream with a given minimum period $T_{4,\text{in}}$ apparently seems to be a valid representation also for some streams that are periodic with jitter. In the next section, we introduce a set of tests that allow the entire problem to be formally captured.

## 5.2 Event Stream Compatibility Tests

We have already formally defined event streams and event models in Chapter 3. An event model is defined as the set of all event streams that share common properties. For instance, the output stream of task $\mathcal{T}_1$ is given *in* the model of periodic events with jitter, i. e. the stream is an element of the model:

$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{\mathsf{P+J}}(T_1, J_1) \in \mathcal{EM}_{\mathsf{P+J}} \tag{5.4}$$

Task $\mathcal{T}_4$ requires a sporadic input stream, the allowed minimum period is further restricted to be at least $T_{4,\text{req}}$. Hence, the acceptable set of input streams is only a *subset* of the model of strictly sporadic events. We call this subset an *event stream requirement* $\mathcal{R}$.

**Definition 5.2.** ***(Event Stream Requirement)***
*An Event Stream Requirement $\mathcal{R}$ is a non-empty set of event streams.*
*A given event stream $\mathcal{S}$ is said to meet a given event stream requirement $\mathcal{R}$, if and only if $\mathcal{S}$ is an element of $\mathcal{R}$.* ■

We formulate the input stream requirement $\mathcal{R}_{4,\text{in}}$ of task $\mathcal{T}_4$ to contain all sporadic streams with a sporadic period of $T_{4,\text{req}}$ and larger:

$$\mathcal{R}_{4,\text{in}} = \left\{\mathcal{S}_{\mathsf{S}}(T) \middle| \, T \geq T_{4,\text{req}}\right\} \subseteq \mathcal{EM}_{\mathsf{S}} \tag{5.5}$$

The entire problem definition is illustrated in Figure 5.3. We said that the periodic output stream with jitter does not meet the sporadic input requirement. Mathematically speaking, the output stream $\mathcal{S}_{1,\text{out}}$ of task $\mathcal{T}_1$ is not included in the input requirement $\mathcal{R}_{4,\text{in}}$ of task $\mathcal{T}_4$:

$$\mathcal{S}_{1,\text{out}} \notin \mathcal{R}_{4,\text{in}} \tag{5.6}$$

Moreover, $\mathcal{S}_{1,\text{out}}$ is not even *in* the right model:

$$\begin{aligned} \mathcal{S}_{1,\text{out}} &\in \mathcal{EM}_{\mathsf{P+J}} \\ \mathcal{S}_{1,\text{out}} &\notin \mathcal{EM}_{\mathsf{S}} \end{aligned} \tag{5.7}$$

The transformed input stream, however, is in the right model:

$$\mathcal{S}_{4,\text{in}} = \mathcal{S}_{\mathsf{S}}(T_1 - J_1) \in \mathcal{EM}_{\mathsf{S}} \tag{5.8}$$

When also the sporadic period $T_{4,\text{in}}$ is within the allowed range, the transformed stream *meets* the target input requirement (see Definition 5.2):

$$\begin{aligned} & \mathcal{S}_{4,\text{in}} \text{ meets } \mathcal{R}_{4,\text{in}} \\ \Leftrightarrow \quad & \mathcal{S}_{4,\text{in}} \in \mathcal{R}_{4,\text{in}} \\ \Leftrightarrow \quad & (\mathcal{S}_{4,\text{in}} \in \mathcal{EM}_{\mathsf{S}}) \wedge (T_{4,\text{in}} \geq T_{4,\text{req}}) \end{aligned} \tag{5.9}$$

The above definitions allow the given situation of the example to be formally captured. Specifically, the *event model (in)compatibility* can be determined using set-containment conditions, such as in Equation 5.6. In the example, we have resolved the incompatibility using an appropriate parameter transformation. However, we have not yet formally verified that the event model interface of Equation 5.3 is correct.

## 5.3 Interface Verification

Event model interfaces must transform a given source stream into a target model representation such, that the possible event timing of the source stream is fully *covered* by the possible timing of the target stream. Otherwise, the scheduling analysis of the target task is not able to consider all event arrival scenarios that can actually occur. We can check this coverage using the four characteristic functions.

**Definition 5.3.** ***(Event Stream Coverage Test)***
*The Event Stream Coverage Test $\mathcal{SC}$ is a function that takes as arguments a source event stream $\mathcal{S}_{source}$ and a target event stream $\mathcal{S}_{target}$ and returns a boolean value.*

*Let $\mathcal{Z}$ be the set of all possible event streams $\mathcal{S}$. Then, $\mathcal{SC}$ is defined by:*

$$\mathcal{SC} : \mathcal{Z} \times \mathcal{Z} \mapsto \{0, 1\} \tag{5.10}$$

*$\mathcal{SC}(\mathcal{S}_{source}, \mathcal{S}_{target})$ returns 1 (true), if the source stream $\mathcal{S}_{source}$ is covered by the target stream $\mathcal{S}_{target}$, and 0 (false) otherwise.*

*A source event stream $\mathcal{S}_{source}$ is covered by a target event stream $\mathcal{S}_{target}$, if and only if all of the following four conditions are true:*

$$\forall\, \Delta t \in \mathbb{R}^+ \quad : \quad \eta^+_{\mathcal{S}_{source}}(\Delta t) \leq \eta^+_{\mathcal{S}_{target}}(\Delta t) \tag{5.11}$$

$$\forall\, \Delta t \in \mathbb{R}^+ \quad : \quad \eta^-_{\mathcal{S}_{source}}(\Delta t) \geq \eta^-_{\mathcal{S}_{target}}(\Delta t) \tag{5.12}$$

$$\forall\, n \in \mathbb{N}^+ \setminus \{0,1\} \quad : \quad \delta^-_{\mathcal{S}_{source}}(n) \geq \delta^-_{\mathcal{S}_{target}}(n) \tag{5.13}$$

$$\forall\, n \in \mathbb{N}^+ \setminus \{0,1\} \quad : \quad \delta^+_{\mathcal{S}_{source}}(n) \leq \delta^+_{\mathcal{S}_{target}}(n) \tag{5.14}$$

■

We shall soon see that the transformation of Equation 5.3 successfully passes this test, and hence represents a correct event model interface. In Section 5.3.2, we will proof that the following equation holds:

$$\forall \mathcal{S}_{\mathsf{P+J}}(T,J), J < T : \mathcal{SC}(\mathcal{S}_{\mathsf{P+J}}(T,J), \mathcal{S}_{\mathsf{P}}(T-J)) = 1 \text{ (true)} \tag{5.15}$$

In addition, we define a *Requirement Coverage Test*, that generalizes the just introduced event stream coverage test such that it is also applicable to an event stream requirement.

**Definition 5.4.** ***(Requirement Coverage Test)***
*The Requirement Coverage Test $\mathcal{RC}$ is a function that takes as arguments a source event stream $\mathcal{S}_{source}$ and a target event stream requirement $\mathcal{R}_{target}$ and returns a boolean value.*

*Let $\mathcal{Z}$ be the set of all possible event streams $\mathcal{S}$, and $\mathbb{P}(\mathcal{Z})$ be its power set. Then, $\mathcal{RC}$ is defined by:*

$$\mathcal{RC} : \mathcal{Z} \times \mathbb{P}(\mathcal{Z}) \mapsto \{0,1\} \tag{5.16}$$

*$\mathcal{RC}(\mathcal{S}_{source}, \mathcal{R}_{target})$ returns 1 (true), if the stream $\mathcal{S}_{source}$ is covered by the requirement $\mathcal{R}_{target}$, and 0 (false) otherwise.*

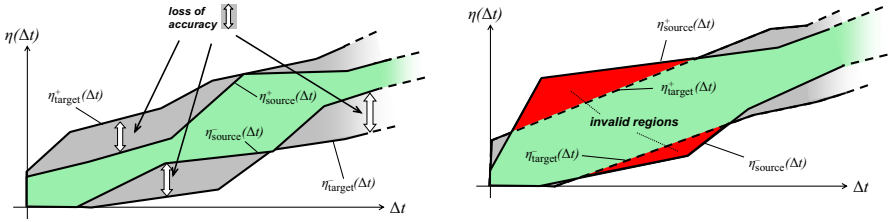
*An event stream $\mathcal{S}_{source}$ is covered by an event stream requirement $\mathcal{R}_{target}$, if and only if the source stream $\mathcal{S}_{source}$ is covered by at least one element of the target requirement:*

$$\mathcal{RC}(\mathcal{S}_{source}, \mathcal{R}_{target}) = \left(\exists \mathcal{S}_{target} \in \mathcal{R}_{target} : (\mathcal{SC}(\mathcal{S}_{source}, \mathcal{S}_{target}) = 1)\right) \tag{5.17}$$

■

This requirement coverage test lets us formally test, if the output stream of task $\mathcal{T}_1$ can be transformed into the model of strictly sporadic events $\mathcal{EM}_{\mathsf{S}}$ at all. This is only possible, if the following equation holds:

$$\mathcal{RC}(\mathcal{S}_{1,\text{out}}, \mathcal{EM}_{\mathsf{S}}) \stackrel{!}{=} 1 \text{ (true)} \tag{5.18}$$




(a) Example of *Covered* Event Streams

(b) Example of *Non-Covered* Event Streams

*Figure 5.4.* Graphical Representation of Stream Coverage

The mentioned coverage tests are applicable to arbitrary streams including numerical streams, for which no abstract properties and parameters are known, e. g. the numerical arrival and service curves that Thiele et al. use [116]. In fact, parameterized event models allow the coverage to be formally checked more efficiently than numerical representations. The characteristic functions, however, are sufficient for the verification of an interface.

### 5.3.1 Graphical Verification

The graphical representation of the characteristic functions illustrates the meaning of stream coverage. Figure 5.4 shows two examples. In Figure 5.4(a), the $\eta$ functions of the source stream reside *between* the corresponding $\eta$ functions of the target stream. More precisely, the $\eta^+_{\text{source}}(\Delta t)$ curve is below the $\eta^+_{\text{target}}(\Delta t)$-curve (condition 5.11 is true), and the $\eta^-_{\text{source}}(\Delta t)$ curve is above the $\eta^-_{\text{target}}(\Delta t)$ curve (condition 5.12 is true). Assuming that also the $\delta$ functions are covered, we can formulate this coverage by:

$$\mathcal{SC}(\mathcal{S}_{\text{source}}, \mathcal{S}_{\text{target}}) = 1(\text{true}) \tag{5.19}$$

Figure 5.4(b) shows a scenario where both conditions 5.11 and 5.12 are violated. The source stream is not covered by the target stream:

$$\mathcal{SC}(\mathcal{S}_{\text{source}}, \mathcal{S}_{\text{target}}) = 0(\text{false}) \tag{5.20}$$

We will now verify the coverage of the event model interface of our initial example. The interface was given by Equation 5.3. Again, we start from the graphical representation, which is given in Figure 5.5. In addition to the integer functions $\eta^+(\Delta t)$ and $\eta^-(\Delta t)$, the "unrounded" versions of these functions ($\tilde{\eta}^+$ and $\tilde{\eta}^-$) are also shown in the figure as dotted lines. In those regions where the $\tilde{\eta}^+$ curve of the target stream is above the $\tilde{\eta}^+$ curve of the source stream, the

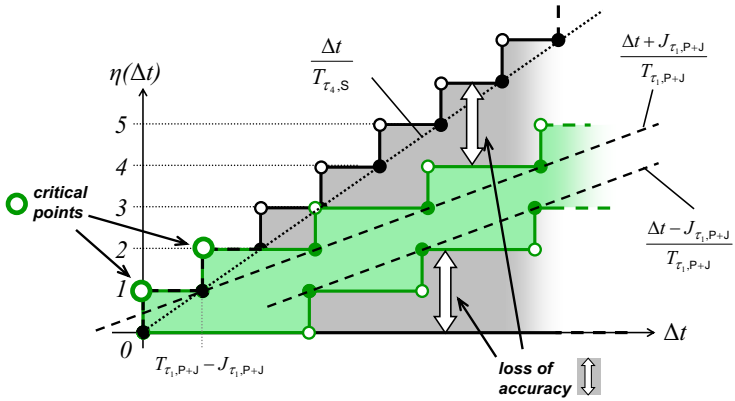



*Figure 5.5.* $\eta(\Delta t)$ Curves of $\mathcal{EMIF}_{\mathsf{P+J}\rightarrow\mathsf{S}}$

$\eta^+$ condition (Equation5.11) is obviously fulfilled. And the regions where the target curve is not above the source curve –the region with the critical points– require only a minimum of additional information. In this case, it is enough to understand that, if the jitter is less than the period, then the two curves intersect at $\Delta t = T_{\mathsf{P+J}} - J_{\mathsf{P+J}} > 0$, and both discrete $\eta^+$ functions have a value of one in that area, which also fulfills the $\eta^+$ condition. The curves furthermore show that also the $\eta^-$ condition of Equation 5.12 is met.

### 5.3.2 Formal Verification

Depending in the involved streams, the formal verification of event stream containment can be complex. In the example, we have the following streams:

$$\begin{aligned} \mathcal{S}_{\text{source}} &= \mathcal{S}_{1,\text{out}} = \mathcal{S}_{\mathsf{P+J}}(T_1, J_1) \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{4,\text{in}} = \mathcal{S}_{\mathsf{S}}(T_4 = T_1 - J_1) \end{aligned}$$

From the previous chapter, we know the characteristic functions of these two streams (in their interval notation):

$$\begin{aligned} \eta^{\mathrm{I}}_{\mathcal{S}_{1,\text{out}}}(\Delta t) &= \left[\max\left(0, \left\lfloor \frac{\Delta t - J_1}{T_1} \right\rfloor\right); \left\lceil \frac{\Delta t + J_1}{T_1} \right\rceil\right] \\ \delta^{\mathrm{I}}_{\mathcal{S}_{1,\text{out}}}(n) &= \left[\max\left(0, (n-1)T_1 - J_1\right); (n-1)T_1 + J_1\right] \\ \eta^{\mathrm{I}}_{\mathcal{S}_{4,\text{in}}}(\Delta t) &= \left[0; \left\lfloor \frac{\Delta t}{T_4} \right\rfloor\right] \\ \delta^{\mathrm{I}}_{\mathcal{S}_{4,\text{in}}}(n) &= [(n-1)T_4; \infty] \end{aligned}$$

#### 5.3.2.1 $\eta^-$ Compatibility Test

Because of the sporadic nature of the target event stream, condition 5.12 can be trivially proved:

$$\eta^-_{1,\text{out}}(\Delta t) = \max\left(0, \left\lfloor \tfrac{\Delta t - J_1}{T_1} \right\rfloor\right) \geq 0 = \eta^-_{4,\text{in}}(\Delta t)$$

q.e.d. (5.21)

#### 5.3.2.2 $\delta^+$ Compatibility Test

Checking test 5.14 is trivial, too:

$$\delta^+_{\mathcal{S}_{1,\text{out}}}(n) = (n-1)T_1 + J_1 \leq \infty = \delta^+_{\mathcal{S}_{4,\text{in}}}(n)$$

q.e.d. (5.22)

#### 5.3.2.3 $\delta^-$ Compatibility Test

Checking condition 5.13 is slightly more complex. Note that $\delta(n)$ is only defined for $n \geq 2$:

$$\begin{aligned}
& n && \geq && 2 && | -1, \times(-J_{1,\text{out}}) \\
\Leftrightarrow\ & -(n-1)J_{1,\text{out}} && \leq && -J_{1,\text{out}} && | +(n-1)T_{1,\text{out}} \\
\Leftrightarrow\ & (n-1)(T_{1,\text{out}} - J_{1,\text{out}}) && \leq && (n-1)T_{1,\text{out}} - J_{1,\text{out}} && \\
\Leftrightarrow\ & (n-1)T_{4,\text{in}} && \leq && (n-1)T_{1,\text{out}} - J_{1,\text{out}} && \\
\Leftrightarrow\ & \delta^-_{4,\text{in}}(n) && \leq && \delta^-_{1,\text{out}}(n) && \\
& \text{q.e.d.} &&&&&&
\end{aligned} \tag{5.23}$$

#### 5.3.2.4 $\eta^+$ Compatibility Test

Checking the $\eta^+$ condition from Equation 5.11 is more difficult. The proof is split into two parts. The first part considers time intervals of size $\Delta t \leq T_{4,\text{in}} = T_{1,\text{out}} - J_{1,\text{out}}$, while the second part treats larger $\Delta t$s. First, we will see that both $\eta^+(\Delta t)$ functions have a value of 1 (one) for $0 < \Delta t \leq T_{1,\text{out}} - J_{1,\text{out}}$. We start with the periodic stream with jitter:

$$\begin{aligned}
& 0 && < && \Delta t && \leq && T_{1,\text{out}} - J_{1,\text{out}} && | + J_{1,\text{out}} \\
\Leftrightarrow\ & 0 < J_{1,\text{out}} && < && \Delta t + J_{1,\text{out}} && \leq && T_{1,\text{out}} && | \times \tfrac{1}{T_{1,\text{out}}}, T_{1,\text{out}} > 0 \\
\Leftrightarrow\ & 0 < \tfrac{J_{1,\text{out}}}{T_{1,\text{out}}} && < && \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} && \leq && 1 && | \lceil\ \rceil \\
\Rightarrow\ & 0 && < && \lceil \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} \rceil && = && 1 && \\
& \Rightarrow && && \eta^+_{1,\text{out}}(\Delta t) && = && 1 && 
\end{aligned} \tag{5.24}$$

Next, we proof that $\eta^+(\Delta t) = 1$ for $0 < \Delta t \leq T_{4,\text{in}} = T_{1,\text{out}} - J_{1,\text{out}}$ on the sporadic event stream:

$$\begin{aligned}
& 0 && < && \Delta t && \leq && T_{1,\text{out}} - J_{1,\text{out}} && | \times \tfrac{1}{T_{1,\text{out}} - J_{1,\text{out}}}, T_{1,\text{out}} > J_{1,\text{out}} > 0 \\
\Leftrightarrow\ & 0 && < && \tfrac{\Delta t}{T_{1,\text{out}} - J_{1,\text{out}}} && \leq && 1 && | \lceil\ \rceil \\
\Rightarrow\ & 0 && < && \lceil \tfrac{\Delta t}{T_{1,\text{out}} - J_{1,\text{out}}} \rceil && = && 1 && \\
\Leftrightarrow\ & 0 && < && \lceil \tfrac{\Delta t}{T_{4,\text{in}}} \rceil && = && 1 && \\
& \Rightarrow && && \eta^+_{4,\text{in}}(\Delta t) && = && 1 &&
\end{aligned} \tag{5.25}$$

From equations 5.24 and 5.25, we have shown that

$$\forall\ \Delta t \leq T_{4,\text{in}} = T_{1,\text{out}} - J_{1,\text{out}} : \eta^+_{1,\text{out}}(\Delta t) = \eta^+_{4,\text{in}}(\Delta t) = 1. \tag{5.26}$$

The substitution

$$\begin{aligned} & \Delta t > T_{1,\text{out}} - J_{1,\text{out}} \\ \Leftrightarrow \quad & \Delta t = T_{1,\text{out}} - J_{1,\text{out}} + x, x > 0 \\ \Leftrightarrow \quad & x = \Delta t - (T_{1,\text{out}} - J_{1,\text{out}}) > 0 \end{aligned} \tag{5.27}$$

is used to show that condition 5.11 is also met for $\Delta t > T_{4,\text{in}} = T_{1,\text{out}} - J_{1,\text{out}}$:

$$\begin{aligned} & J_{1,\text{out}} > 0 && | -J_{1,\text{out}} \\ \Leftrightarrow \quad & 0 > -J_{1,\text{out}} && | +T_{1,\text{out}} \\ \Leftrightarrow \quad & T_{1,\text{out}} > T_{1,\text{out}} - J_{1,\text{out}} && |()^{-1} \\ \Leftrightarrow \quad & \tfrac{1}{T_{1,\text{out}}} < \tfrac{1}{T_{1,\text{out}} - J_{1,\text{out}}} && | \times x, x > 0 \\ \Leftrightarrow \quad & \tfrac{x}{T_{1,\text{out}}} < \tfrac{x}{T_{1,\text{out}} - J_{1,\text{out}}} && | +1 \\ \Leftrightarrow \quad & 1 + \tfrac{x}{T_{1,\text{out}}} < 1 + \tfrac{x}{T_{1,\text{out}} - J_{1,\text{out}}} && | x = \Delta t - (T_{1,\text{out}} - J_{1,\text{out}}) > 0 \\ \Leftrightarrow \quad & 1 + \tfrac{\Delta t - (T_{1,\text{out}} - J_{1,\text{out}})}{T_{1,\text{out}}} < 1 + \tfrac{\Delta t - (T_{1,\text{out}} - J_{1,\text{out}})}{T_{1,\text{out}} - J_{1,\text{out}}} \\ \Leftrightarrow \quad & 1 + \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} - \tfrac{(T_{1,\text{out}})}{T_{1,\text{out}}} < 1 + \tfrac{\Delta t}{T_{1,\text{out}} - J_{1,\text{out}}} - \tfrac{(T_{1,\text{out}} - J_{1,\text{out}})}{T_{1,\text{out}} - J_{1,\text{out}}} \\ \Leftrightarrow \quad & 1 + \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} - 1 < 1 + \tfrac{\Delta t}{T_{1,\text{out}} - J_{1,\text{out}}} - 1 \\ \Leftrightarrow \quad & \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} < \tfrac{\Delta t}{T_{1,\text{out}} - J_{1,\text{out}}} && |\lceil\rceil \\ \Rightarrow \quad & \lceil \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} \rceil \leq \lceil \tfrac{\Delta t}{T_{1,\text{out}} - J_{1,\text{out}}} \rceil \\ \Leftrightarrow \quad & \lceil \tfrac{\Delta t + J_{1,\text{out}}}{T_{1,\text{out}}} \rceil \leq \lceil \tfrac{\Delta t}{T_{4,\text{in}}} \rceil \\ \Rightarrow \quad & \eta^+_{1,\text{out}}(\Delta t) \leq \eta^+_{4,\text{in}}(\Delta t) \end{aligned} \tag{5.28}$$

Finally, equations 5.26 and 5.28 provide:

$$\forall\, \Delta t > 0 : \eta^+_{1,\text{out}}(\Delta t) \leq \eta^+_{4,\text{in}}(\Delta t)$$

q.e.d. (5.29)

With all four conditions met, the event stream coverage test of Section 5.3 returns 1 (true), and the transformation of Equation 5.1 in fact provides a valid $\mathcal{EMIF}$ for the transformation from a periodic stream with jitter into a sporadic stream without jitter. However, we have seen that the formal proofs can be complex. Where reasonable, we shall use intuitive and comprehensible *graphical proofs*, as presented in Section 5.3.1, rather than complex mathematical equations.

### 5.3.3 Interface Quality

In addition to event stream coverage, there are two other interesting properties of that event model interface, which Figure 5.5 nicely reveals. First, we see that for large $\Delta t$, the curves *diverge*, i. e. the distance between the upper [lower] bound curves of the source and the target stream increases. This is because the single parameter of the target model of strictly sporadic events is not able to capture all details of the source stream (periodic with jitter), but can only conservatively approximate or bound them at the cost of *lost accuracy*, indicated by the white arrows. We call such interfaces *lossy interfaces*.

An interface that transforms a source stream into a target stream *without* loss of accuracy is called *lossless*. Lossless transformation requires the characteristic functions –and thus the curves– of the source and the target stream to be *identical*. We can use the coverage test to check if an interface is lossless, since the coverage test must be successful *in both directions*, otherwise the characteristic functions would differ:

$$\begin{aligned} &\mathcal{EMIF}_{\text{source}\rightarrow\text{target}} \text{ is lossless} \Leftrightarrow \\ &\quad \mathcal{SC}(\mathcal{S}_{\text{source}}, \mathcal{S}_{\text{target}}) = \mathcal{SC}(\mathcal{S}_{\text{target}}, \mathcal{S}_{\text{source}}) = 1 \end{aligned} \tag{5.30}$$

Likewise, we can check for *lossiness* using the following test:

$$\mathcal{EMIF}_{\text{source}\rightarrow\text{target}} \text{ is lossy} \Leftrightarrow \mathcal{SC}(\mathcal{S}_{\text{target}}, \mathcal{S}_{\text{source}}) = 0 \tag{5.31}$$

Second, we look at small $\Delta t$s, especially the two *critical points* where the curves are identical. This illustrates the *tightness* of the transformation in Equation 5.3. Any smaller value for $T_{4,\text{in}}$ would violate the $\eta^+$-condition in this critical region, while larger values would result in further accuracy loss, even for $\Delta t < T_{1,\text{out}} - J_{1,\text{out}}$. In other words, the transformation is –although lossy– optimal in the sense that no other valid transformation with less accuracy loss can be found.

We can also formalize an optimality test. We are looking for the interfaces that yield the tightest target stream. Let $\mathcal{EMIF}_1$ and $\mathcal{EMIF}_2$ be two event model interfaces that transform a source stream $\mathcal{S}_{\text{source}}$ into the target streams $\mathcal{S}_{\text{target},1}$ and $\mathcal{S}_{\text{target},2}$, respectively. We assume that both interfaces are correct, i. e. the event stream coverage test from Definition 5.3 is successful:

$$\mathcal{SC}(\mathcal{S}_{\text{source}}, \mathcal{S}_{\text{target},1}) = 1(\text{true}) \wedge \mathcal{SC}(\mathcal{S}_{\text{source}}, \mathcal{S}_{\text{target},2}) = 1(\text{true}) \tag{5.32}$$

If the target stream $\mathcal{S}_{\text{target},1}$ is covered by the target stream $\mathcal{S}_{\text{target},2}$, then the interface $\mathcal{EMIF}_1$ is *tighter* that $\mathcal{EMIF}_2$:

$$\begin{aligned} &\mathcal{EMIF}_1 \text{ is tighter than } \mathcal{EMIF}_2 \\ &\quad \Leftrightarrow \mathcal{SC}(\mathcal{S}_{\text{target},1}, \mathcal{S}_{\text{target},2}) = 1(\text{true}) \end{aligned} \tag{5.33}$$

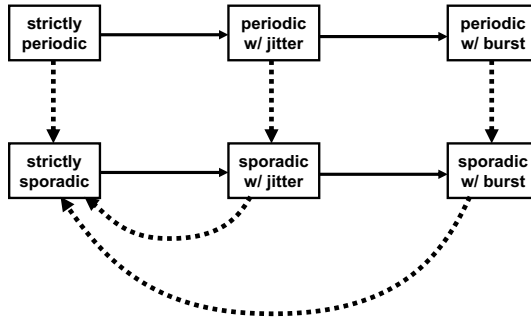



*Figure 5.6.* Existing Atomic Event Model Interfaces

Finally, an interface $\mathcal{EMIF}_{\text{opt}}$ is optimal, if and only if it is at least as tight as (and thus covers) any other interface $\mathcal{EMIF}_X$ for the same event stream combination:

$$\begin{aligned}&\mathcal{EMIF}_{\text{opt,source}\rightarrow\text{target}} \text{ is optimal}\\ &\quad\Leftrightarrow \forall \mathcal{EMIF}_{X,\text{source}\rightarrow\text{target}} \neq \mathcal{EMIF}_{\text{opt,source}\rightarrow\text{target}}:\\ &\qquad\qquad \mathcal{SC}(\mathcal{S}_{\text{target,opt}}, \mathcal{S}_{\text{target,X}}) = 1(\text{true})\end{aligned} \tag{5.34}$$

We will see that for all interface-able model combinations within the six-class model set, a single optimal interface can be found.

## 5.4 Existing Interfaces

Mathematically speaking, finding an $\mathcal{EMIF}$ for a specific combination of event models ($\mathcal{EM}_{\text{source}} \rightarrow \mathcal{EM}_{\text{target}}$) equals solving Equation 5.2 such that for *at least some* $\mathcal{S}_{\text{source}} \in \mathcal{EM}_{\text{source}}$, the requirement coverage test is successful:

$$\mathcal{RC}(\mathcal{S}_{\text{source}}, \mathcal{EM}_{\text{target}}) = 1 \text{ (true)} \tag{5.35}$$

If this test is not successful for *all* streams in the source event model, then the domain of the interface must be constrained accordingly. The event model interface from the introductory example (see Equation 5.3) is an example for an interface with a constrained domain. Furthermore, we are looking for optimal interfaces with respect to Equation 5.34.

Due to the non-linearities in the characteristic functions, a formal procedure can be complex. In most cases, the empirical approach is more promising. We have seen in the example, that the graphical representations of the characteristic functions provides a comprehensible illustration of the problem. In particular the continuous $\tilde{\eta}$ functions are well suited to supplement textually

motivated model transformations, so that detailed mathematical derivations are reduced to a minimum. Correctness and optimality proofs are provided where necessary.

We will now summarize the existing event model interfaces with respect to the six-class model set. We shall start with *atomic interfaces*, shown in Figure 5.6. Lossless interfaces are illustrated by straight lines, lossy ones by dotted lines in that figure. Later, we will compose other interfaces from these atomic ones. Our initial example from periodic with jitter into strictly sporadic is an example for a non-atomic interface.

## 5.5 Lossless Event Model Interfaces

Lossless interfaces exist only for the transformations already investigated in the preceding chapter. We saw that execution and scheduling adds distortion or non-determinism to task outputs. So, we needed output models that support this additional expressive power (modeling capabilities) compared to the input models. In other words, the output models are *more general* than the input models, and a lossless transformation is possible.

### 5.5.1 Strictly Periodic → Periodic with Jitter

The transformation itself is trivial. The period of the source stream is preserved, and the jitter in the target stream is zero:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P}\rightarrow\mathsf{P+J}} \quad &: \quad \mathcal{EM}_{\mathsf{P}} \mapsto \mathcal{EM}_{\mathsf{P+J}} \\ \mathcal{S}_{\text{target}} \quad &= \quad \mathcal{S}_{\mathsf{P+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = 0) \end{aligned} \tag{5.36}$$

The proofs are trivial. With the $J$ being 0 (zero), the characteristic functions of the periodic with jitter target model (see Section 3.5) are reduced to the functions of the periodic stream (see Section 3.3). This transformation is lossless because the corresponding characteristic functions are identical, captured by the following equation:

$$\mathcal{SC}(\mathcal{S}_{\mathsf{P}}(T), \mathcal{S}_{\mathsf{P+J}}(T, 0)) = \mathcal{SC}(\mathcal{S}_{\mathsf{P+J}}(T, 0), \mathcal{S}_{\mathsf{P}}(T)) = 1 \tag{5.37}$$

### 5.5.2 Periodic with Jitter → Periodic with Burst

The same idea applies to this transformation. The period and the jitter of the source stream is preserved, and the minimum distance is set accordingly. This reduces the characteristic functions of the newly defined burst model (see Section 4.3) into those of the jitter model (see Section 3.5):

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P+J}\rightarrow\mathsf{P+B}} \quad &: \quad \mathcal{EM}_{\mathsf{P+J}} \mapsto \mathcal{EM}_{\mathsf{P+B}} \\ \mathcal{S}_{\text{target}} \quad &= \quad \mathcal{S}_{\mathsf{P+B}} \left( \begin{array}{l} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = J_{\text{source}}, \\ d_{\text{target}} = \max(0, T_{\text{source}} - J_{\text{source}}) \end{array} \right) \end{aligned} \tag{5.38}$$

This interface is lossless because Equation 5.30 is true.

### 5.5.3 Strictly Sporadic → Sporadic with Jitter

The systematic orthogonalization of jitter on the one hand, and the duality between sporadic and periodic models on the other hand, has an important practical impact. Each lossless transformation between standard event models in the periodic domain has its equivalent transformation in the sporadic domain. So, this $\mathsf{S} \rightarrow \mathsf{S{+}J}$ transformation is identical to the equivalent transformation in the periodic domain ($\mathsf{P} \rightarrow \mathsf{P{+}J}$), mentioned in Section 5.5.1.

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{S}\rightarrow\mathsf{S+J}} &: \mathcal{EM}_{\mathsf{S}} \mapsto \mathcal{EM}_{\mathsf{S+J}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = 0) \end{aligned} \tag{5.39}$$

### 5.5.4 Sporadic with Jitter → Sporadic with Burst

Again, this is just the sporadic version of the transformation from periodic with jitter into periodic with burst. The equivalent $\mathcal{EMIF}$ applies:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{S+J}\rightarrow\mathsf{S+B}} &: \mathcal{EM}_{\mathsf{S+J}} \mapsto \mathcal{EM}_{\mathsf{S+B}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+B}} \left( \begin{array}{l} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = J_{\text{source}}, \\ d_{\text{target}} = \max(0, T_{\text{source}} - J_{\text{source}}) \end{array} \right) \end{aligned} \tag{5.40}$$

### 5.5.5 Model Reductions

We have seen that a strictly periodic event stream can be modeled as a periodic stream with a zero jitter without accuracy loss. Practically speaking, a zero jitter can be considered as "no jitter". That means the stream actually has the same characteristic functions as a strictly periodic stream without jitter. This allows the event model interface of Equation 5.36 to be inverted, i. e. the target and source stream can be exchanged, as already indicated by Equation 5.30. We call such transformations *model reductions*. They differ from general event model interfaces in that their domain is restricted to special cases, e. g. a zero jitter. Similarly, we can transform a burst stream into a jittery stream, if and only if the minimum distance between events in the original burst stream fulfills the requirements of the jitter models. Both reductions are applicable to periodic and sporadic streams.

#### Reduction $\mathcal{EMIF}$s

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P+J}\overset{\text{red}}{\rightarrow}\mathsf{P}} &: \{\mathcal{S}_{\mathsf{P+J}}(T, J) \in \mathcal{EM}_{\mathsf{P+J}} | J = 0\} \mapsto \mathcal{EM}_{\mathsf{P}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{P}}(T_{\text{target}} = T_{\text{source}}) \end{aligned} \tag{5.41}$$

$$\begin{aligned}\mathcal{EMIF}_{\mathsf{P+B}\overset{\text{red}}{\rightarrow}\mathsf{P+J}} \quad &: \quad \{\mathcal{S}_{\mathsf{P+B}}(T,J,d) \in \mathcal{EM}_{\mathsf{P+B}} | d = \max(0, T-J)\} \\ &\qquad\qquad \mapsto \mathcal{EM}_{\mathsf{P}} \\ \mathcal{S}_{\text{target}} \quad &= \quad \mathcal{S}_{\mathsf{P+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = J_{\text{source}}) \end{aligned} \tag{5.42}$$

$$\begin{aligned}\mathcal{EMIF}_{\mathsf{S+J}\overset{\text{red}}{\rightarrow}\mathsf{S}} \quad &: \quad \{\mathcal{S}_{\mathsf{S+J}}(T,J) \in \mathcal{EM}_{\mathsf{S+J}} | J = 0\} \mapsto \mathcal{EM}_{\mathsf{S}} \\ \mathcal{S}_{\text{target}} \quad &= \quad \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = T_{\text{source}}) \end{aligned} \tag{5.43}$$

$$\begin{aligned}\mathcal{EMIF}_{\mathsf{S+B}\overset{\text{red}}{\rightarrow}\mathsf{S+J}} \quad &: \quad \{\mathcal{S}_{\mathsf{S+B}}(T,J,d) \in \mathcal{EM}_{\mathsf{S+B}} | d = \max(0, T-J)\} \\ &\qquad\qquad \mapsto \mathcal{EM}_{\mathsf{P}} \\ \mathcal{S}_{\text{target}} \quad &= \quad \mathcal{S}_{\mathsf{S+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = J_{\text{source}}) \end{aligned} \tag{5.44}$$

Such model reductions are always lossless, just like their lossless inverse transformations. But they are only applicable to special case situations such as a zero jitter. In practice, there is no difference between the event timing in a strictly periodic event stream and a stream with a zero jitter. It is merely a matter of modeling that becomes important in an automatic interfacing procedure, where parameters must be well-defined. In Sections 5.6.2 and 5.6.3, we will see also lossy transformations from sporadic with jitter and burst into the model of strictly sporadic events. Therefore, we explicitly indicate a reduction interface by "$\overset{\text{red}}{\rightarrow}$" (see the $\mathcal{EMIF}$ equations above).

## 5.6 Lossy Event Model Interfaces

Besides the lossless transformations, there is a set of lossy transformations with slightly different properties.

### 5.6.1 Transforming Periodic into Sporadic Models

As Figure 5.6 shows, there are three lossy transformations from a periodic model into its sporadic counterpart. Compared to the aforementioned lossless interfaces, these transformations follow the second systematic concept of the six-class model set, the concept of conditional event production and unknown task behavior that has been discussed in Section 4.5. Figure 5.7 shows the curves of the transformation of periodic with jitter into sporadic with jitter which nicely illustrates the accuracy loss. The corresponding interfaces are straight-forward:

$$\begin{aligned}\mathcal{EMIF}_{\mathsf{P}\rightarrow\mathsf{S}} \quad &: \quad \mathcal{EM}_{\mathsf{P}} \mapsto \mathcal{EM}_{\mathsf{S}} \\ \mathcal{S}_{\text{target}} \quad &= \quad \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = T_{\text{source}}) \end{aligned} \tag{5.45}$$

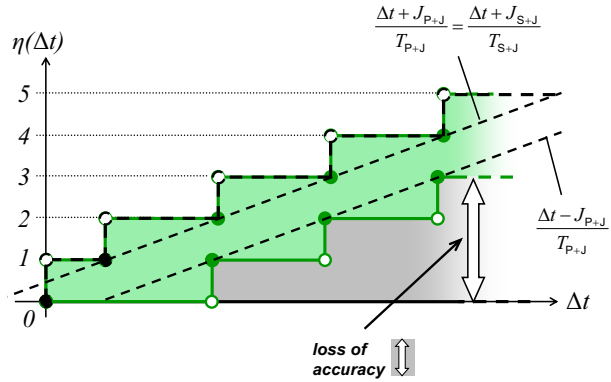



*Figure 5.7.* $\eta(\Delta t)$ Curves of $\mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+J}}$

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+J}} &: \mathcal{EM}_{\mathsf{P+J}} \mapsto \mathcal{EM}_{\mathsf{S+J}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+J}} \left( \begin{array}{l} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = J_{\text{source}} \end{array} \right) \end{aligned} \tag{5.46}$$

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+B}} &: \mathcal{EM}_{\mathsf{P+B}} \mapsto \mathcal{EM}_{\mathsf{S+B}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+B}} \left( \begin{array}{l} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = J_{\text{source}}, \\ d_{\text{target}} = d_{\text{source}} \end{array} \right) \end{aligned} \tag{5.47}$$

In each of the three transformations, the parameters of source and target model are identical, so also the upper bound arrival function and the minimum distance of the transformation's source and target model are identical. This clearly fulfills Equations 5.11 and 5.13. Only the lower bound event arrival functions $\eta^-(\Delta t)$ [upper-bound distance functions $\delta^+(n)$] differ, since these are implicitly zero [infinity] for the sporadic target model. But Equations 5.12 and 5.14 are fulfilled, so the interfaces provide valid transformations.

The fact that they are lossy is easily seen in Figure 5.7, and can by mathematically checked using Equation 5.31.

### 5.6.2 Sporadic with Jitter → Strictly Sporadic

There are two more lossy atomic interfaces. The first one, shown in Figure 5.8, transforms sporadic events with jitter into the model of strictly sporadic events. Because of the systematic model structure, we can re-use much

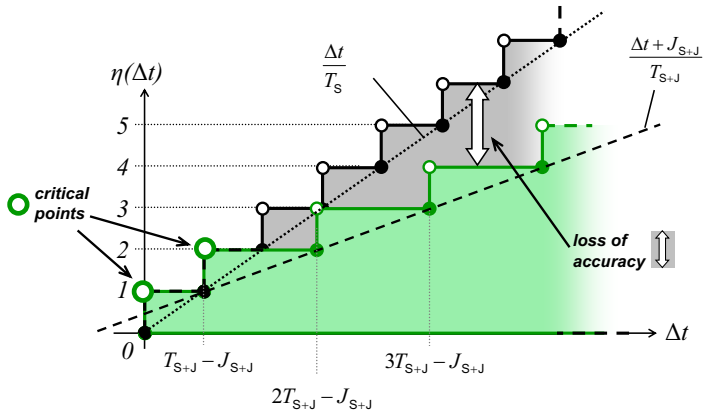



*Figure 5.8.* $\eta(\Delta t)$ Curves of $\mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S}}$

of the work from the introductory example, where we transformed a periodic stream with jitter into a sporadic stream:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S}} &: \{\mathcal{S}_{\mathsf{S+J}}(T,J) \in \mathcal{EM}_{\mathsf{S+J}} | J < T\} \mapsto \mathcal{EM}_{\mathsf{S}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = T_{\text{source}} - J_{\text{source}}) \end{aligned} \tag{5.48}$$

We can directly re-use the relatively complex proofs of the $\eta^+$- and $\delta^-$-conditions from Sections 5.3.2.4 and 5.3.2.2. The remaining proofs are trivial:

$$\eta^-_{\mathsf{S+J}}(\Delta t) = 0 \quad \geq \quad 0 = \eta^-_{\mathsf{S}}(\Delta t)$$
$$\text{q.e.d.} \tag{5.49}$$

$$\delta^+_{\mathsf{S+J}}(n) = \infty \quad \leq \quad \infty = \delta^+_{\mathsf{S}}(n)$$
$$\text{q.e.d.} \tag{5.50}$$

All four compatibility conditions from Section 5.3 are fulfilled.

### 5.6.3 Sporadic with Burst → Strictly Sporadic

Figure 5.9 shows the $\eta$ curves of this transformation which is relatively straight-forward. The sought-after parameter of the sporadic model (the sporadic period $T_{\mathsf{S}}$) represents a minimum inter-arrival time. In the model of bursty events, this is directly given by the minimum distance $d_{\mathsf{S+B}}$, but essen-

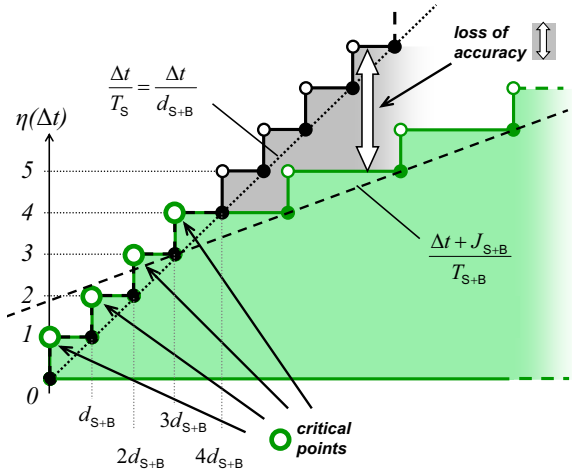



*Figure 5.9.* $\eta(\Delta t)$ Curves of $\mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S}}$

tially must be larger than zero to avoid infinite event frequencies:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S}} &: \{\mathcal{S}_{\mathsf{S+B}}(T,J,d) \in \mathcal{EM}_{\mathsf{S+J}} | d > 0\} \mapsto \mathcal{EM}_{\mathsf{S}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = d_{\text{source}}) \end{aligned} \tag{5.51}$$

Without formal proofs, we can easily see in Figure 5.9 that the target model just captures the transient frequency during bursts to fulfill the $\eta^+$ condition in the critical points. When the burst region is over, the target curve is above the source curve and both curves diverge. Hence, the transformation is lossy.

All lossy interfaces share a key property. They transform a more detailed and sometimes more constrained source stream into a more general target model. This comes at the cost of lost accuracy. While the transformations from periodic models into their sporadic counterparts preserve the details of the upper-bound arrival functions $\eta^+$ (and the minimum distance $\delta^-$), the lower-bound arrival functions $\eta^-$ were –very simplifying– set to zero. In effect, the maximum system load remains constant, the transformation just adds non-determinism to the model. In contrast, the other two mentioned transformations reduce only accuracy in the upper-bound arrival function (and the minimum distance function). This results in higher average frequencies and hence higher system load an analysis must assume. We have already identified this phenomenon as a modeling insufficiency in Section 4.4.3.

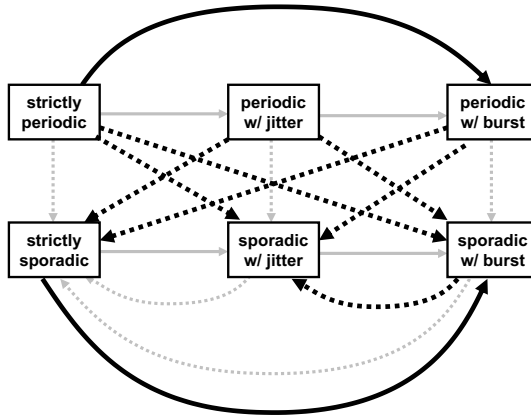



*Figure 5.10.* Existing Composite Event Model Interfaces

## 5.7 Composite Event Model Interfaces

The atomic event model interfaces introduced thus far can be concatenated to compose a number of additional lossless and lossy interfaces. Mathematically, the transformation functions are composed:

$$\begin{aligned}
\mathcal{S}_{\text{target}} &= \mathcal{EMIF}_{\text{source}\rightarrow\text{target}}\left(\mathcal{S}_{\text{source}}\right) \\
&= \underbrace{\left(\mathcal{EMIF}_{\text{intermediate}\rightarrow\text{target}} \circ \mathcal{EMIF}_{\text{source}\rightarrow\text{intermediate}}\right)}_{\text{function composition}}\left(\mathcal{S}_{\text{source}}\right) \\
&= \mathcal{EMIF}_{\text{intermediate}\rightarrow\text{target}}(\underbrace{\mathcal{EMIF}_{\text{source}\rightarrow\text{intermediate}}\left(\mathcal{S}_{\text{source}}\right)}_{\mathcal{S}_{\text{intermediate}}}) \\
&= \mathcal{EMIF}_{\text{intermediate}\rightarrow\text{target}}\left(\mathcal{S}_{\text{intermediate}}\right)
\end{aligned} \tag{5.52}$$

Figure 5.10 gives an overview of the existing composite interfaces, which are summarized briefly in the following paragraphs.

### 5.7.1 Strictly Periodic → Periodic with Burst

This transformation can be composed of the transformation from strictly periodic into periodic with jitter (see Equation 5.36), and the transformation

from periodic with jitter into periodic with burst (see Equation 5.38):

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P}\to\mathsf{P+B}} &: \mathcal{EM}_{\mathsf{P}} \mapsto \mathcal{EM}_{\mathsf{P+B}} \\ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{P+B}} &= \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{P+B}} \circ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{P+J}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{P+B}} \begin{pmatrix} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = 0, \\ d_{\text{target}} = T_{\text{source}} \end{pmatrix} \end{aligned} \tag{5.53}$$

### 5.7.2 Strictly Sporadic → Sporadic with Burst

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+B}} &: \mathcal{EM}_{\mathsf{S}} \mapsto \mathcal{EM}_{\mathsf{S+B}} \\ \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+B}} &= \mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S+B}} \circ \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+J}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{P+B}} \begin{pmatrix} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = 0, \\ d_{\text{target}} = T_{\text{source}} \end{pmatrix} \end{aligned} \tag{5.54}$$

Equation 5.54 shows the sporadic counterpart of the previous composition in the periodic domain. The resulting interface is, of course, lossless, too. There are no other lossless interfaces.

### 5.7.3 Strictly Periodic → Sporadic with Jitter

This transformation can be achieved as a two-step composition in two ways, each possibility represents a different path from the source to the target model. The two possible paths are illustrated in Figure 5.11. One possibility has a periodic stream with jitter as an intermediate model, the other one first transforms the source stream into the model of strictly sporadic events:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{P}\to\mathsf{S+J}} &: \mathcal{EM}_{\mathsf{P}} \mapsto \mathcal{EM}_{\mathsf{S+J}} \\ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{S+J}} &= \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+J}} \circ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{P+J}} \\ &= \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+J}} \circ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{S}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = 0) \end{aligned} \tag{5.55}$$

It is not really surprising that both possibilities yield the same composed interface. In Section 4.7, we mentioned the orthogonality of the two axes. The horizontal axis where the jitter increases and finally leads to transient bursts, and the vertical axis with its transformation from periodic into sporadic models. Both of the non-atomic transformations above are composed of atomic ones from both of these axes. The concept of jitter has been added, and a transformation from periodic to sporadic has been made. The transformation order independency just underlines the observed orthogonality.

Furthermore, it is obvious that the composed transformation is lossy since both include the lossy transformation from periodic into sporadic. In general,

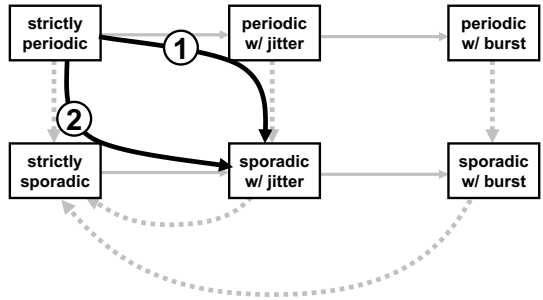



*Figure 5.11.* Two Paths from Strictly Periodic to Sporadic with Jitter

a composite transformation is lossy if at least one of its atomic transformations is lossy.

### 5.7.4 Strictly Periodic → Sporadic with Burst

In this case we have two atomic transformations on the horizontal axis and one on the vertical axis. Again, the result is independent of the order in which the corresponding atomic transformations are applied:

$$\begin{aligned}
\mathcal{EMIF}_{\mathsf{P}\to\mathsf{S+B}} &: \mathcal{EM}_{\mathsf{P}} \mapsto \mathcal{EM}_{\mathsf{S+B}} \\
\mathcal{EMIF}_{\mathsf{P}\to\mathsf{S+B}} &= \mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+B}} \circ \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{P+B}} \circ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{P+J}} \\
&= \mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S+B}} \circ \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+J}} \circ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{P+J}} \\
&= \mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S+B}} \circ \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+J}} \circ \mathcal{EMIF}_{\mathsf{P}\to\mathsf{S}} \\
\mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+B}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = 0, d_{\text{target}} = T_{\text{source}})
\end{aligned} \tag{5.56}$$

### 5.7.5 Periodic with Jitter → Strictly Sporadic

This is our initial example for which only one possible composition exists. We first transform the periodic stream with jitter into the model of sporadic events with jitter (see Sec. 5.6.1), and then further into the model of strictly sporadic events (see Sec. 5.6.2). This concatenation can be mathematically expressed by:

$$\begin{aligned}
\mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S}} &: \{\mathcal{S}_{\mathsf{P+J}}(T, J) \in \mathcal{EM}_{\mathsf{P+J}} | J < T\} \mapsto \mathcal{EM}_{\mathsf{S}} \\
\mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S}} &= \mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S}} \circ \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+J}} \\
\mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = T_{\text{source}} - J_{\text{source}})
\end{aligned}$$

### 5.7.6 Periodic with Jitter → Sporadic with Burst

On both possible paths, one via periodic with burst and another via sporadic with jitter, the resulting composite $\mathcal{EMIF}$ is:

$$\begin{aligned}
\mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+B}} &: \mathcal{EM}_{\mathsf{P+J}} \mapsto \mathcal{EM}_{\mathsf{S+B}} \\
\mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+B}} &= \mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+B}} \circ \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{P+B}} \\
&= \mathcal{EMIF}_{\mathsf{S+J}\to\mathsf{S+B}} \circ \mathcal{EMIF}_{\mathsf{P+J}\to\mathsf{S+J}} \\
\mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+J}} \left( \begin{array}{l} T_{\text{target}} = T_{\text{source}}, \\ J_{\text{target}} = J_{\text{source}}, \\ d_{\text{target}} = \max(0, T_{\text{source}} - J_{\text{source}}) \end{array} \right)
\end{aligned} \tag{5.57}$$

### 5.7.7 Periodic with Burst → Strictly Sporadic

The only possible path is through sporadic with burst:

$$\begin{aligned}
\mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S}} &: \{\mathcal{S}_{\mathsf{P+B}}(T, J, d) \in \mathcal{EM}_{\mathsf{P+B}} | d > 0\} \mapsto \mathcal{EM}_{\mathsf{S}} \\
\mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S}} &= \mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S}} \circ \mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+B}} \\
\mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = d_{\text{source}})
\end{aligned} \tag{5.58}$$

### 5.7.8 Periodic with Burst → Sporadic with Jitter

This interface is obtained by concatenating the interfaces $\mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+B}}$, $\mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S}}$, and $\mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+J}}$:

$$\begin{aligned}
\mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+J}} &: \mathcal{EM}_{\mathsf{P+B}} \mapsto \mathcal{EM}_{\mathsf{S+J}} \\
\mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+J}} &= \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+J}} \circ \mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S}} \circ \mathcal{EMIF}_{\mathsf{P+B}\to\mathsf{S+B}} \\
\mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = J_{\text{source}})
\end{aligned} \tag{5.59}$$

### 5.7.9 Sporadic with Burst → Sporadic with Jitter

Here, two atomic interfaces are required:

$$\begin{aligned}
\mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S+J}} &: \mathcal{EM}_{\mathsf{S+B}} \mapsto \mathcal{EM}_{\mathsf{S+J}} \\
\mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S+J}} &= \mathcal{EMIF}_{\mathsf{S}\to\mathsf{S+J}} \circ \mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{S}} \\
\mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S+J}}(T_{\text{target}} = T_{\text{source}}, J_{\text{target}} = J_{\text{source}})
\end{aligned} \tag{5.60}$$

## 5.8 Including Sporadically Periodic Events

After presenting event model interfaces for all possibly transformations within the six-class model set, we will now show how the model of sporadically periodic events, as defined by Audsley and Tindell [4, 121], can be transformed into this six-class model, and vice versa. We have explicitly excluded their model from the six-class model because of its inability to efficiently handle jitter as a key influence of scheduling and execution which we mentioned in

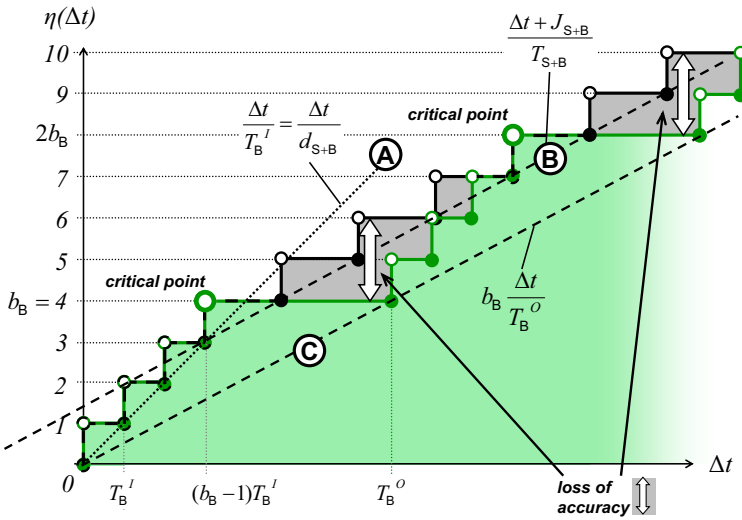



*Figure 5.12.* $\eta(\Delta t)$ Curves of $\mathcal{EMIF}_{\mathsf{B}\rightarrow\mathsf{S+B}}$

Section 4.2.3. But despite its weaknesses in capturing outputs, the model has proven very valuable for specific input descriptions.

To no surprise, the model of sporadically periodic events shares key concepts with the newly introduced model of sporadic events with burst. Even though both models are based on fundamentally different observations, they describe a generally periodic event stream as an upper bound for long-term observations, and additionally bound the maximum transient event frequencies. So, both have a certain notion of bursts. In order to avoid confusion when dealing with these two models, we will refer to the model of sporadically periodic events as "Tindell's burst model", indicated by the index B.

In the following sections, we concentrate on transformations between these two models. More interfaces can be obtained using the idea of interface composition (see Section 5.7) and are not explicitly mentioned further.

### 5.8.1 Tindell's Burst → Sporadic with Burst

Figure 5.12 illustrates the $\eta(\Delta t)$ curves of this transformation. The $\eta^-$ condition is trivially met because of the sporadic nature of both models. To meet also the $\eta^+$ condition, we have to derive the period $T_{\mathsf{S+B}}$, the jitter $J_{\mathsf{S+B}}$, and the minimum distance $d_{\mathsf{S+B}}$ for the model of sporadic events with burst. The three straight lines labeled A, B, and C represent three continuous $\tilde{\eta}^+$ functions

that help in finding the interface. Line A captures the timing behavior of both curves during the initial burst, line B represents the long-term behavior of the sporadic model with bursts, and line C captures the repetition rate of "Tindell bursts".

For small $\Delta t$s, the minimum inter-arrival time dominates the worst-case event arrival of $\eta^+(\Delta t)$. Line A in Figure 5.12 illustrates that both $\eta^+$ curves are identical. It is relatively obvious that the minimum distance $d_{\mathsf{S+B}}$ of the target model is exactly the inner period $T^I_{\mathsf{B}}$ of Tindell's burst model:

$$d_{\mathsf{S+B}} = T^I_{\mathsf{B}} \tag{5.61}$$

In order to find the other parameters, we concentrate on two critical points in the curves of Figure 5.12. The left-most critical point at

$$\Delta t_{\text{crit,1}} = (b_{\mathsf{B}} - 1)T^I_{\mathsf{B}} \tag{5.62}$$

captures the end of the first "Tindell burst". In the example, the burst length equals four ($b = 4$). The second critical point marks the end of the second Tindell burst, and other critical points appear at the end of each burst, but they are not shown. We need to choose the period and the jitter of the sporadic event stream with bursts such that all critical points of Tindell's burst model are covered.

Apparently, this is guaranteed when lines B and C are parallel, and line A meets line B at $\Delta t_{\text{crit,1}}$, given by Equation 5.62. We start by deriving the period $T_{\mathsf{S+B}}$, which equals the average maximum frequency of the events in Tindell's burst model:

$$T_{\mathsf{S+B}} = \frac{T^O_{\mathsf{B}}}{b_{\mathsf{B}}} \tag{5.63}$$

Now, we can derive the remaining jitter parameter that determines the horizontal displacement of line B with respect to line C. We known that lines A and B must meet at $\Delta t_{\text{crit,1}} = (b_{\mathsf{B}} - 1)T^I_{\mathsf{B}}$. At that point, the $\eta^+(\Delta t)$ functions that describe the lines must be equal. We resolve the equation to obtain the jitter parameter:

$$\begin{aligned}
& \eta^+_{\mathsf{B}}(\Delta t_{\text{crit,1}}) &&= \eta^+_{\mathsf{S+B}}(\Delta t_{\text{crit,1}}) \\
\Leftrightarrow \quad & \eta^+_{\mathsf{B}}((b_{\mathsf{B}} - 1)T^I_{\mathsf{B}}) &&= \eta^+_{\mathsf{S+B}}((b_{\mathsf{B}} - 1)T^I_{\mathsf{B}}) \\
\Leftrightarrow \quad & (b_{\mathsf{B}} - 1)T^I_{\mathsf{B}} + J_{\mathsf{S+B}} &&= (b_{\mathsf{B}} - 1)T_{\mathsf{S+B}} \qquad \text{|Equation 5.63} \\
\Leftrightarrow \quad & J_{\mathsf{S+B}} &&= (1 - \tfrac{1}{b_{\mathsf{B}}})T^O_{\mathsf{B}} - (b_{\mathsf{B}} - 1)T^I_{\mathsf{B}}
\end{aligned} \tag{5.64}$$

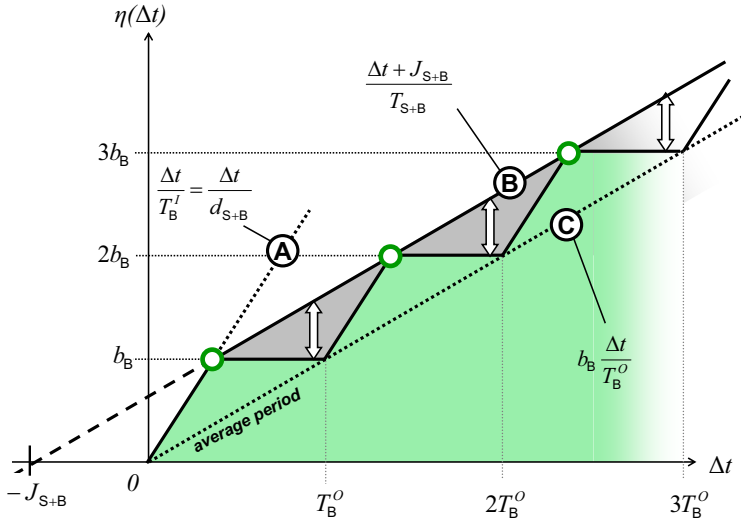



*Figure 5.13.* An Abstract View of the Curves of $\mathcal{EMIF}_{\mathsf{B}\to\mathsf{S+B}}$

We finally combine Equations 5.61, 5.63, and 5.64 to obtain the event model interface:

$$\mathcal{EMIF}_{\mathsf{B}\to\mathsf{S+B}} : \mathcal{EM}_{\mathsf{B}} \mapsto \mathcal{EM}_{\mathsf{S+B}}$$
$$\mathcal{S}_{\text{target}} = \mathcal{S}_{\mathsf{S+B}} \left( \begin{array}{l} T_{\text{target}} = \frac{T^O_{\text{source}}}{b_{\text{source}}}, \\ J_{\text{target}} = (1 - \frac{1}{b_{\text{source}}}) T^O_{\text{source}} - (b_{\text{source}} - 1) T^I_{\text{source}}, \\ d_{\text{target}} = T^I_{\text{source}} \end{array} \right) \tag{5.65}$$

This transformation from Tindell's burst model into the new model of sporadic events with burst ($\mathsf{B} \to \mathsf{S+B}$) is lossy. This can be easily seen in Figure 5.12. But this kind of lossiness is quite different from the other lossy transformations within the six-class models. Those transformation could in fact very tightly capture the stream behavior form small $\Delta t$ but suffered an increasing inaccuracy with increasing $\Delta t$, either or the $\eta^-$ side, as for the transition from periodic into sporadic (see Section 5.6.1), or in the $\eta^+$ side, as for the transformation from sporadic with burst into strictly sporadic (see Section 5.6.3).

The interface $\mathcal{EMIF}_{\mathsf{B}\to\mathsf{S+B}}$ has, interestingly, a bounded long-term accuracy loss. In some points in time, specifically all critical points which we have

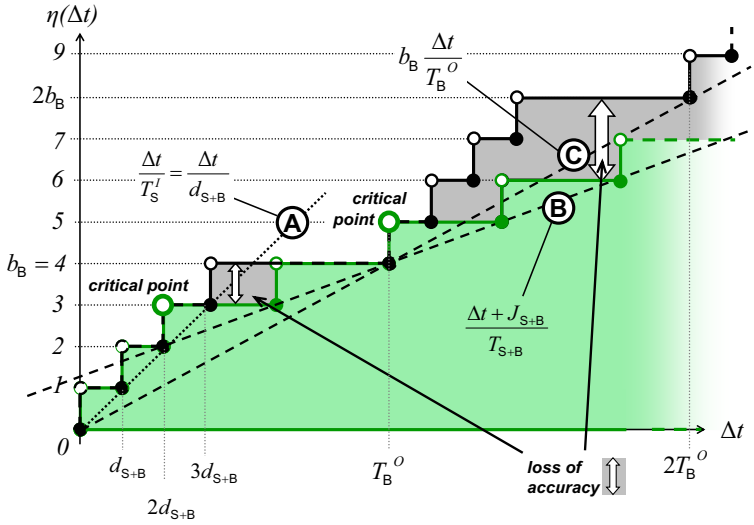



*Figure 5.14.* $\eta(\Delta t)$ Curves of $\mathcal{EMIF}_{\mathsf{S+B}\rightarrow\mathsf{B}}$

mentioned to reappear at the end of every Tindell burst, the functions have even identical values. Figure 5.13 provides a more abstract view on the continuous $\tilde{\eta}$ functions of this transformation. We can see that the sawtooth shaped curve of the source stream regularly meets the linear jitter curve of the target stream. This bounded accuracy loss shows that the newly introduced model of sporadic events with burst is not overly inaccurate when capturing Tindell's burst model.

### 5.8.2 Sporadic with Burst → Tindell's Burst

Figure 5.14 illustrates the details of the transformation from the model of sporadic events with burst into Tindell's burst model. One parameter, the inner period $T_{\mathsf{B}}^{I}$ of Tindell's burst model, is found quite trivial; it equals the minimum distance $d_{\mathsf{S+B}}$ of the sporadic event model with burst:

$$T_{\mathsf{B}}^{I} = d_{\mathsf{S+B}} \tag{5.66}$$

This guarantees that the maximum transient frequencies are bounded, illustrated by line A in the figure.

The derivation of the other two parameters, the burst length $b_{\mathsf{B}}$ and the outer period $T_{\mathsf{B}}^{O}$, is considerably more complex. Figure 5.14 highlights two critical points. The first one is determined by the end of the burst of the sporadic stream

with burst, or the time where the transient burst turns into a periodic stream with a large jitter. We have thoroughly discussed such behavior in Section 4.3. According to Equation 4.15, this happens at time $\Delta t_{\text{crit},1}$:

$$\Delta t_{\text{crit},1} = \frac{J_{\mathsf{S+B}}\, d_{\mathsf{S+B}}}{T_{\mathsf{S+B}} - d_{\mathsf{S+B}}} \tag{5.67}$$

It is quite clear that the burst of the target stream must cover this source burst. In other words, the sought-after burst length of the target $b_{\mathsf{B}}$ must not be smaller than the number of events at the aforementioned point in time $\Delta t_{\text{crit},1}$. This way, we can formulate a lower bound for $b_{\mathsf{B}}$:

$$\begin{aligned} b_{\mathsf{B}} &\geq \eta^{+}_{\mathsf{S+B}}(\Delta t_{\text{crit},1}) \\ b_{\mathsf{B}} &\geq \eta^{+}_{\mathsf{S+B}}\left(\frac{J_{\mathsf{S+B}}\, d_{\mathsf{S+B}}}{T_{\mathsf{S+B}} - d_{\mathsf{S+B}}}\right) \\ b_{\mathsf{B}} &\geq \left\lceil \frac{J_{\mathsf{S+B}}}{T_{\mathsf{S+B}} - d_{\mathsf{S+B}}} \right\rceil \end{aligned} \tag{5.68}$$

The second critical point in Figure 5.14 captures the beginning of the second Tindell burst in the target stream, at $\Delta t_{\text{crit},2}$. At this time, the $\eta^{+}$ functions of both source and target streams increase from $b_{\mathsf{B}}$ to $b_{\mathsf{B}} + 1$. In other words, $\Delta t_{\text{crit},2}$ equals the minimum distance between $b_{\mathsf{B}} + 1$ events in both streams:

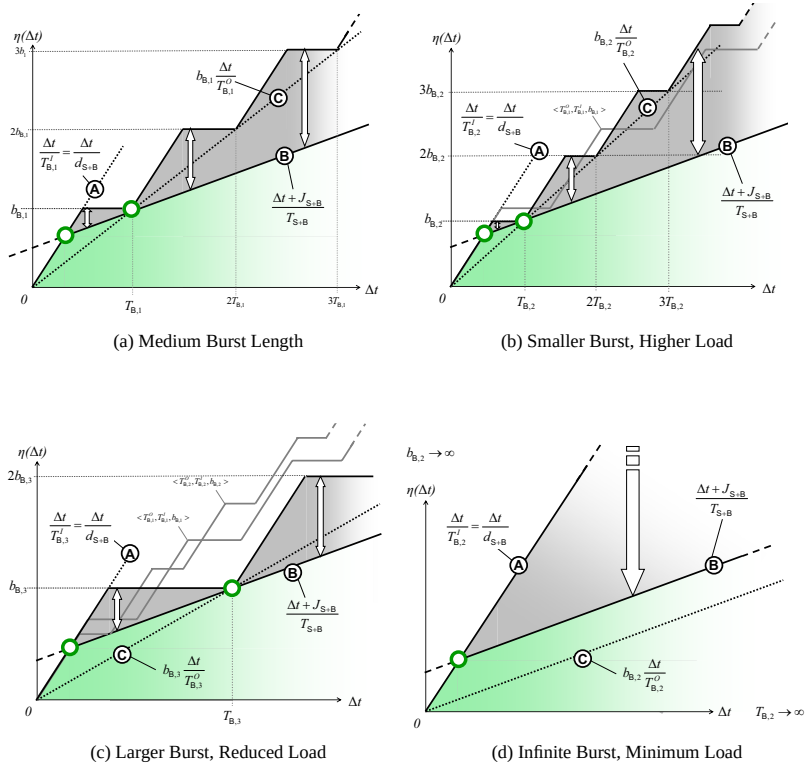
$$\Delta t_{\text{crit},2} = \delta^{-}_{\mathsf{S+B}}(b_{\mathsf{B}} + 1) \stackrel{!}{=} \delta^{-}_{\mathsf{B}}(b_{\mathsf{B}} + 1) \tag{5.69}$$

We apply the minimum event distance functions $\delta^{-}(n)$ to obtain the outer period $T^{O}_{\mathsf{B}}$:

$$\begin{aligned} \delta^{-}_{\mathsf{B}}(b_{\mathsf{B}} + 1) &= \delta^{-}_{\mathsf{S+B}}(b_{\mathsf{B}} + 1) \\ T^{O}_{\mathsf{B}} &= ((b_{\mathsf{B}} + 1) - 1)\, T_{\mathsf{S+B}} - J_{\mathsf{S+B}} \\ &= b_{\mathsf{B}} T_{\mathsf{S+B}} - J_{\mathsf{S+B}} \end{aligned} \tag{5.70}$$

Although this equation provides a linear relation between the two remaining parameters $b_{\mathsf{B}}$ and $T^{O}_{\mathsf{B}}$, neither of the values can be explicitly determined. We known only a lower bound for Tindell's burst length parameter $b_{\mathsf{B}}$, given by Equation 5.68. As other information is not available, there is a certain amount of freedom how the interface can be parameterized. We can choose any burst length $b_{\mathsf{B}}$ that fulfills Equation 5.68, and the outer period $T^{O}_{\mathsf{B}}$ is then obtained from the linear relation given by Equation 5.70. With respect to the curve representation, that means that we can choose the second critical point to be anywhere on the $\eta^{+}$ curve of the source stream (line B).

We illustrate this by a qualitative comparison between four different parameter sets. Figure 5.15 shows the $\eta$ curves of four different $\mathcal{EMIF}_{\mathsf{S+B}\rightarrow\mathsf{B}}$. All




(a) Medium Burst Length

(b) Smaller Burst, Higher Load

(c) Larger Burst, Reduced Load

(d) Infinite Burst, Minimum Load

*Figure 5.15.* Comparison of Four $\mathcal{EMIF}_{\mathsf{S+B}\to\mathsf{B}}$ Choices

interfaces share the same source stream but they differ in their specific values for $b_{\mathsf{B}}$ and $T_{\mathsf{B}}^{O}$. We see that, in contrast to the transformation in the opposite direction, the accuracy loss generally increases with increasing $\Delta t$ for all four curves. Depending on the chosen interface parameters $b_{\mathsf{B}}$ and $T_{\mathsf{B}}^{O}$, the accuracy loss, however, can have considerably different characteristics.

Figure 5.15(a) shows a reference interface with an arbitrary (but valid) choice of $b_{\mathsf{B}}$ and $T_{\mathsf{B}}^{O}$. Figure 5.15(b) has a smaller burst length $b_{\mathsf{B}}$. In effect, the accuracy loss *between* the two critical points in the figure is less compared to the first interface in Figure 5.15(a). However, this optimization for small $\Delta t$ comes at the cost of larger accuracy loss for larger $\Delta t$ beyond the second critical point. Basically, a small burst length results in a small outer period, rep-

resenting a higher long-term load. The gray curve represents the $\eta^+$ function from the reference example and allows a direct comparison.

An example with a larger burst length is shown in Figure 5.15(c). Again, we have included the curves of the two previous examples as gray curves to allow quick comparison. We see that the larger burst length $b_{\mathrm{B}}$ leads to an increasing transient accuracy loss for small $\Delta t$ (between the two critical points). On the other hand, the long-term loss is reduced. Figure 5.15(c) finally contains a rather theoretical but interesting paradoxical case. The average load is reduced to a minimum, the curves B and C are parallel. To achieve this, we would need to set the burst length to infinity, which basically contradicts the initial objective of load reduction.

We see that, while each realistic interface is "better" than others for specific regions of $\Delta t$, none will be optimal over all others for all $\Delta t$s. On the contrary, an infinite set of Pareto-optimal interfaces exist. For a given source stream $\mathcal{S}_{\text{source}}$, two Pareto-optimal interfaces have target streams $\mathcal{S}_{\text{target,1}}$ and $\mathcal{S}_{\text{target,2}}$, which do not fully contain each other:

$$\mathcal{SC}(\mathcal{S}_{\text{target,1}}, \mathcal{S}_{\text{target,2}}) = 0(\text{false}) \wedge \mathcal{SC}(\mathcal{S}_{\text{target,2}}, \mathcal{S}_{\text{target,1}}) = 0(\text{false}) \quad (5.71)$$

This enables a trade-off between short-term transient accuracy loss versus long-term average loss. The actual choice could possibly be influenced by other, external objectives such as a schedule length (which would bound the "interesting" regions for $\Delta t$) or a latency constraint, and are not considered further here.

## 5.9 Summary

We have introduced a set of definitions to formally capture the event stream interfacing problem in the presence of multiple different event streams with different characteristics. The curve representation of the characteristic functions of event streams provide a very comprehensible illustration of the interfacing step. Several tests enable us to check if a given source stream can be transformed to meet a specific target requirement, and if this transformation can be performed without accuracy loss. The availability of parameterized models allows an efficient, formal interfacing procedure. The event stream coverage test, however, does not essentially require parameterized models, it also applies to situations in which only the event curves are known. Consequently, it is generally possible to capture *any* event stream with our six-class model set, as long as the stream is (or can be) described as a upper- and lower-bound event curves, including the numerical curves from Thiele's work [116] and Gresser's event vectors [39].

We have derived eighteen event model interfaces *within* the six-class model. When we also consider the six situations of identical source and target models, where no transformation is actually required, we can now solve two thirds

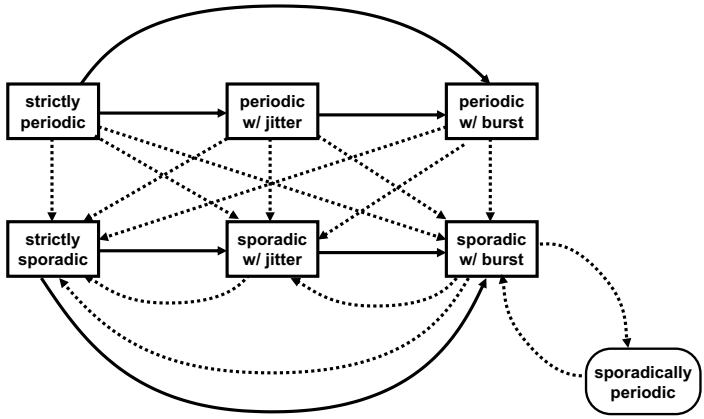



*Figure 5.16.* All Existing Event Model Interfaces

(24 out of 36) of the theoretically possible event model combinations. Some of these interfaces impose specific restrictions to the source stream parameters. For instance, the minimum event distance must be larger than zero when transforming an event stream with burst into a strictly sporadic stream.

Finally, we have presented transformations between the model of sporadic events with burst and Tindell's burst model in both directions. This also shows that event model interfacing is not limited to the newly introduced six-class model set, and that it is generally possibly to include other event models in this interfacing procedure, with totally different parameters. All introduced interfaces are illustrated in Figure 5.16. Solid lines represent lossless interfaces, while dotted lines illustrate lossy ones.

Chapter 6

# EVENT ADAPTATION FUNCTIONS

In the previous chapter, we introduced the concept of event model interfaces to transform event streams given in one event model, into another event model. These interfaces represent mathematical model transformations. We defined a set of compatibility tests that these transformations must fulfill, and derived the transformations for a set of event model combinations. We could see that there are, however, situations for which no plain event model interface exist. These are those situations, which fail the requirement coverage test of Definition 5.4.

This chapter introduces *Event Adaptation Functions* ($\mathcal{EAF}$) that change the event stream timing of a given source stream such that the target requirement can be met. In contrast to event model interfaces that are introduced for modeling and analysis purposes, adaptation functions represent additional design elements, usually buffers and timers, that must be implemented.

An introductory example first illustrates the problem and allows us to formally define the adaptation problem. Then, we derive adaptation functions for those model combinations, for which a plain $\mathcal{EMIF}$ does not exist. We will see that *traffic shaping* that is known from network design [27] can be used to modify the timing of events to comply with given constraints. We distinguish between *periodic shaping* and *sporadic shaping*. Finally, we investigate the properties of *polling* and its influence on input and output event streams .

## 6.1 Introductory Example

Recall the example of Figure 5.1. A producer task $\mathcal{T}_1$ outputs an event stream that does not meet the input requirement of the receiver task $\mathcal{T}_4$. The sender produces a periodic event stream with jitter (see Equation 5.4):

$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{\mathsf{P+J}}(T_1, J_1) \in \mathcal{EM}_{\mathsf{P+J}}$$

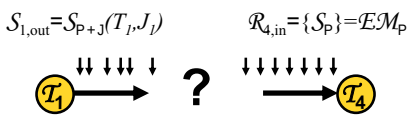



*Figure 6.1.* Introductory Example

But in contrast to the example from the previous chapter, the receiver task $\mathcal{T}_4$ now requires strictly periodic input events:

$$\mathcal{R}_{4,\text{in}} = \mathcal{EM}_{\mathsf{P}} \tag{6.1}$$

The problem formulation is illustrated in Figure 6.1. Again, the models are incompatible:

$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{\mathsf{P+J}}(T_1, J_1) \notin \mathcal{R}_{4,\text{in}} = \mathcal{EM}_{\mathsf{P}} \tag{6.2}$$

This time, the requirement coverage test from Definition 5.4 fails:

$$\mathcal{RC}(\mathcal{S}_{\text{source}}, \mathcal{R}_{4,\text{in}}) = 0(\text{false}) \tag{6.3}$$

We can easily prove this by showing that one of the conditions (Equations 5.11 to 5.14) is false. But before that we need to make a few assumptions. It is obvious that, if an event model interface exists, the transformations target stream would have the same period as the source stream:

$$\mathcal{S}_{\text{target}} = \mathcal{S}_{4,\text{in}} = \mathcal{S}_{\mathsf{P}}(T_{1,\text{out}}) \tag{6.4}$$

We now choose $\eta^+(\Delta t)$ and claim that the $\eta^+$ condition of Equation 5.11 is not fulfilled:

$$\begin{aligned}
& J_1 > 0 && | + \Delta t \\
\Leftrightarrow \quad & \Delta t + J_1 > \Delta t && | \tfrac{1}{T_1}, T_1 > 0 \\
\Leftrightarrow \quad & \tfrac{\Delta t + J_1}{T_1} > \tfrac{\Delta t}{T_1} && \\
\Leftrightarrow \quad & \tfrac{\Delta t + J_1}{T_1} \not\leq \tfrac{\Delta t}{T_1} && \\
\Leftrightarrow \quad & \eta^+_{\text{source}}(\Delta t) \not\leq \eta^+_{\text{source}}(\Delta t) &&
\end{aligned} \tag{6.5}$$

q.e.d.

This is sufficient to show that no plain event model interface exists. In other words, an event stream parameter transformation that resolves this situation does not exist, and we need another approach. Basically, a jitter can be seen as *distortion* that has been (accidentally) added to a initially periodic stream and must be eliminated now. Jitter elimination is commonly performed by re-synchronizing the stream to its original period.

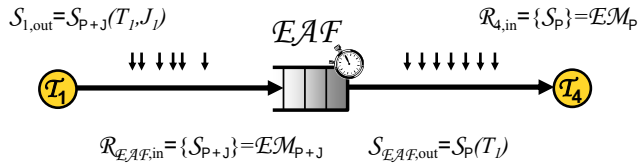



*Figure 6.2.* Application of Event Adaptation Functions $\mathcal{EAF}$

## 6.2 Periodic Synchronization

Traffic shaping uses *buffering* and *time-triggering* and delays events in order to generate some specific communication timing characteristics. All incoming events are temporarily stored for a certain amount of time to obtain the required output characteristics. Figure 6.2 shows the example with an additional FIFO buffer between the two known application tasks. This shaper accepts any periodic stream with jitter at its input, and the output stream of task $\mathcal{T}_1$ meets this requirement:

$$\mathcal{S}_{1,\text{out}} = \mathcal{S}_{\mathsf{P+J}}(T_1, J_1) \in \mathcal{R}_{\mathcal{EAF},\text{in}} = \mathcal{EM}_{\mathsf{P+J}} \tag{6.6}$$

At its output, the shaper produces a periodic stream with the input stream's original period (see Equation 6.4), thereby fulfilling the input requirement of user task $\mathcal{T}_4$:

$$\mathcal{S}_{\mathcal{EAF},\text{out}} = \mathcal{S}_{\mathsf{P}}(T_1) \in \mathcal{R}_{4,\text{in}} = \mathcal{EM}_{\mathsf{P}} \tag{6.7}$$

### 6.2.1 Shaper Implementation

Shaping requires appropriate buffer memory and some means to control the timing. A shaper can be implemented on the same resource as the sending or the receiving task. The local memory that is usually connected to the CPU can be used to temporarily store the (data) events. Often, a CPU has peripheral timer units connected or on-chip (micro-controller) that can be used for controlling the output timing. This way, the shaper becomes (part of) the communication driver of the run-time system. A shaper can also be implemented as a dedicated system function using additional design elements such as a small micro-controller or a dedicated HW block in an ASIC with a small memory and a timer. This thesis is not intended to theorize about the actual implementation but focuses on the properties of shapers and their implementation-independent impact on the system-level scheduling and analysis task. In this context, we need to consider that periodic shapers have their own internal timing and a local buffer. We are interested in a) the required buffer size to dimension (and

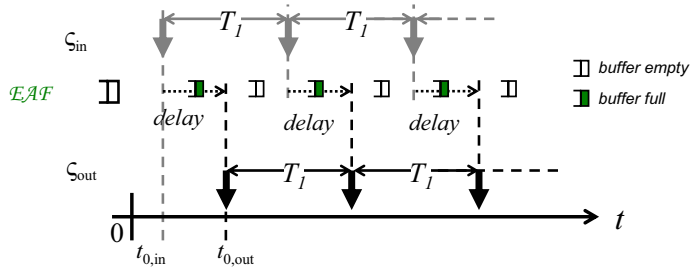



*Figure 6.3.* Buffering of Strictly Periodic Events

optimize) system memory, and b) the buffering delay which adds to path latencies and contributes to the system timing properties.

### 6.2.2 Shaper Properties

We start with applying periodic shaping to a periodic stream with a zero jitter, i. e. to a strictly periodic stream. In this case, the output of $\mathcal{T}_1$ could potentially be $\mathcal{EMIF}$-transformed into a strictly periodic stream (see Section 5.5.5) that meets the input requirement of $\mathcal{T}_4$. But a shaper between both tasks considerably changes the situation.

We mentioned that a shaper is usually implemented with its own internal timer which might not be synchronized with either the senders or the receivers input-output timing. This makes the situation more complex than in case of a plain $\mathcal{EMIF}$.

Figure 6.3 illustrates a strictly periodic shaping scenario. Two event sequences are shown. The sequence $\varsigma_{\text{in}}$ presents the input to the shaper coming from $\mathcal{T}_1$, and the shapers output $\varsigma_{\text{out}}$ becomes the new input of task $\mathcal{T}_4$. Both sequences are identical except a constant non-zero phase difference, or offset difference $t_{0,\text{out}} - t_{0,\text{in}}$. This phase delay exists because of unsynchronized timers.

A buffer of size one seems to be sufficient in the example. However, it is limited to those situations, where the shaper starts outputting events *after* the first event has been produced by task $\mathcal{T}_1$ (case 1). This must not necessarily be the case, because –again– the two tasks are not temporally correlated. Figure 6.4 illustrates another situation (case 2), where the shaper outputs the first event *earlier* than the sender ($t_{0,\text{out}} < t_{0,\text{in}}$). In this case, we need to pre-load the buffer to avoid buffer underrun.

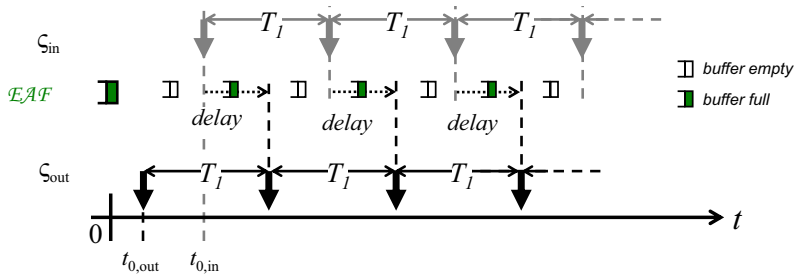



*Figure 6.4.* Avoiding Buffer Underrun by Pre-loaded Buffer

### 6.2.3 The Conservative Approach

We see that the two cases require a different buffer set-up. If we know in advance which the system behavior will be, we use the corresponding set-up for the buffer. However, in most cases, it is not clear which task –sender or shaper– will be earlier than the other, so we need to conservatively set up the buffer to guarantee correct behavior in both cases 1 and 2. Hence, we will need one pre-loaded (data) event to avoid buffer underrun, and *another* free buffer to avoid overflow. That means the buffer must be able to store two events.

This has consequences for the minimum and maximum delay in case 1 (early input). Compared to the scenario in Figure 6.3, the initial pre-loaded event results in an *additional delay* of exactly one period, illustrated in Figure 6.5. Similarly, the buffer of size two represents an *over-dimensioning* in case 2 (early output). Although it does not affect the event timing, 50 percent of the buffer memory seems wasted. Figure 6.6 shows a scenario where the second buffer is never used.

Such seemingly unnecessary over-dimensioning is not an artificial result from the proposed shaping approach. Rather, the conservative approach is the only solution in practice and cannot be optimized in situations, when no specific information about the exact requirements of the communication, i. e. a timer synchronization, is available. And this is very often the case when subsystems are integrated without considering all side effects in detail. In other cases, the amount of pre-loaded data in a FIFO buffer might be constrained by the application. Signal processing applications, for instance, use buffer pre-load to implement specific functional details such as recursive filter functions of a certain degree [68], based on the theory of synchronous data-flow (SDF) [67].

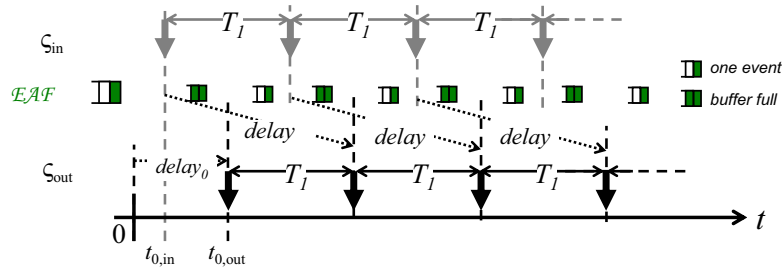



*Figure 6.5.* Additional Delay

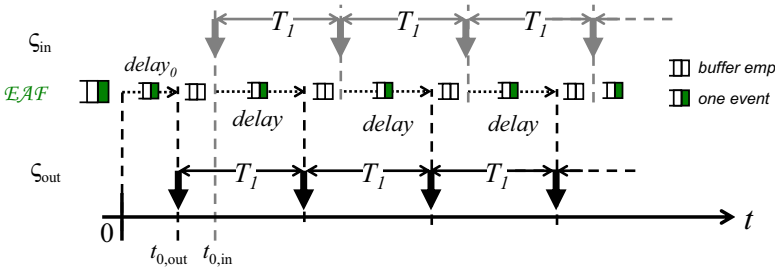



*Figure 6.6.* Over-dimensioning of Buffer Size

In all situations, the conservative design avoids subtle design pitfalls, since accidentally disregarding either case 1 or 2 could potentially result in memory over- or underflow and/or an unexpected temporal (and possibly functional) behavior.

### 6.2.4 Corner-Case Analysis

In the previous chapter, we have seen that the characteristic curves provide an excellent starting point for formal considerations of event model interfacing. The curves also help in formalizing the idea and challenges of event adaptation, because they are well suited to illustrate the properties of the corner-case scenarios 1 and 2.

We start with case 2, where event input was assumed late and output early. Figure 6.7 shows the corner-case event curves: the as-late-as-possible shapers

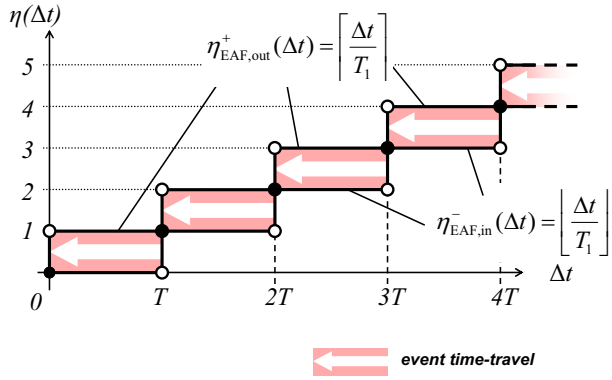



*Figure 6.7.* Event Input and Output Curves of Buffer Underrun Situation

input curve, given by the lower-bound event arrival function

$$\eta^{-}_{\mathcal{EAF},\mathrm{in}}(\Delta t) = \left\lfloor \frac{\Delta t}{T_1} \right\rfloor ,$$

and the as-early-as-possible shaper output curve, given by

$$\eta^{+}_{\mathcal{EAF},\mathrm{out}}(\Delta t) = \left\lceil \frac{\Delta t}{T_1} \right\rceil .$$

This curve representation reveals exactly the aforementioned problem of buffer underrun. We see that the output curve $\eta^{+}_{\mathcal{EAF},\mathrm{out}}(\Delta t)$ is (except for integer multiples of the period) above the input curves, i. e. there are more events outputted than arrived. The arrows in Figure 6.7 illustrate that the events would need to *time-travel*, an apparently impossible solution.

#### 6.2.4.1 Avoiding Buffer Underrun

The pre-loaded events move the curve up. We therefore introduce the notion of *available* events, which are the input events *plus* the pre-loaded ones:

$$\eta_{\mathcal{EAF},\mathrm{av}}(\Delta t) = \eta_{\mathcal{EAF},\mathrm{in}}(\Delta t) + 1 \tag{6.8}$$

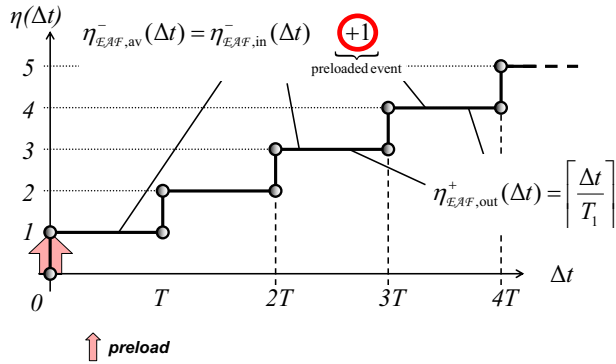



*Figure 6.8.* Case 2: Event Availability and Output Curves

Now, we must make sure that there are always enough events available for being outputted to prevent buffer underrun. This can be easily proven:

$$
\begin{array}{rrcl}
 & \left\{ \begin{array}{ll} 0 & \text{if } \frac{\Delta t}{T_1} \in \mathbb{N} \\ 1 & \text{otherwise} \end{array} \right\} & = & \left\lceil \frac{\Delta t}{T_1} \right\rceil - \left\lfloor \frac{\Delta t}{T_1} \right\rfloor \\
\Rightarrow & 1 & \geq & \left\lceil \frac{\Delta t}{T_1} \right\rceil - \left\lfloor \frac{\Delta t}{T_1} \right\rfloor \quad \mid + \left\lfloor \frac{\Delta t}{T_1} \right\rfloor \\
\Leftrightarrow & \left\lfloor \frac{\Delta t}{T_1} \right\rfloor + 1 & \geq & \left\lceil \frac{\Delta t}{T_1} \right\rceil \\
\Leftrightarrow & \eta^-_{\mathcal{EAF},\text{in}}(\Delta t) + 1 & \geq & \eta^+_{\mathcal{EAF},\text{out}}(\Delta t) \\
\Leftrightarrow & \eta^-_{\mathcal{EAF},\text{av}}(\Delta t) & \geq & \eta^+_{\mathcal{EAF},\text{out}}(\Delta t)
\end{array}
$$

q.e.d.

Figure 6.8 shows the corresponding curves, which are identical except for integer multiples of the period. The lower-bound availability function $\eta^-_{\mathcal{EAF},\text{av}}$ is right-continuous, while the upper-bound output function $\eta^+_{\mathcal{EAF},\text{out}}$ is left-continuous at these points; more a mathematical detail than a useful property in practice. In this situation, all events experience a zero buffering delay, and the buffer does just not underrun.

#### 6.2.4.2 Avoiding Buffer Overflow

In case 1, the input events arrive before the corresponding output events. The worst-case event curves for this scenario are: the as-soon-as-possible

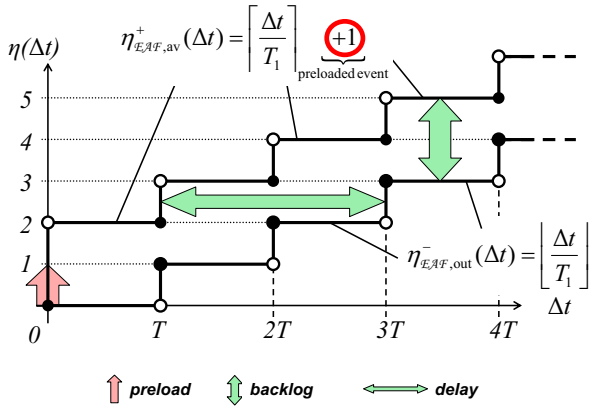



*Figure 6.9.* Case 1: Event Availability and Output Curves

event input curve and the as-late-as-possible event output curve. Figure 6.9 shows both curves. We are looking for the maximum required buffer size and the maximum delay. The two arrows between the curves directly provide the sought-after information. The *horizontal distance* between the curves captures the relative time between the arrival and release of a buffered event, and hence represents the input-output delay of the $\mathcal{EAF}$. And the *vertical distance* provides the difference between the number of events arrived and released within a given period of time $\Delta t$, and hence the number of events stored in the buffer over $\Delta t$.

### 6.2.5 Formal Shaper Analysis

Cruz formalized this idea as *network calculus* [27]. He applied the approach to the horizontal and vertical distances between continuous network traffic arrival curves and network node service curves to obtain the servicing *delay* and the buffering *backlog*. Later, Boudec [15] and Thiele et. al. [116] used corresponding methods to analyze scheduling. According to Cruz [27], the backlog can be calculated by:

$$\mathit{backlog}_{\mathcal{EAF}} = \sup_{\Delta t > 0} \{\eta_{\mathcal{EAF},\mathrm{av}}(\Delta t) - \eta_{\mathcal{EAF},\mathrm{out}}(\Delta t)\} \tag{6.9}$$

Calculating the delay is usually more complex. Cruz [27] provided no calculations but only a general set property:

$$\mathit{delay}_{\mathcal{EAF}} = \sup_{\Delta t>0}\left\{\inf_{\tau>0}\left\{\eta_{\mathcal{EAF},\text{in}}(\Delta t) \leq \eta_{\mathcal{EAF},\text{out}}(\Delta t+\tau)\right\}\right\} \tag{6.10}$$

Both properties depend on the characteristic functions $\eta(\Delta t)$ and $\delta(n)$ of the involved curves. And since these functions return intervals with a lower and an upper bound, the backlog and delay calculation must consider this, i. e. we will also determine minimum and maximum values, requiring to carefully consider minimum and maximum values inside the $\sup$ and $\inf$ functions, or their discrete versions $\min$ and $\max$, respectively.

#### 6.2.5.1 Buffering Backlog

The maximum backlog captures the number of events that wait for being actually outputted to the consumer task $\mathcal{T}_4$. The maximum backlog thus represents the required shaper buffer size:

$$\begin{aligned}
\mathit{backlog}^{+}_{\mathcal{EAF}} &= \sup_{\Delta t>0}\left\{\eta^{+}_{\mathcal{EAF},\text{av}}(\Delta t) - \eta^{-}_{\mathcal{EAF},\text{out}}(\Delta t)\right\} \\
&= \max_{\Delta t>0}\left\{\left\lceil\frac{\Delta t}{T_1}\right\rceil + 1 - \left\lfloor\frac{\Delta t}{T_1}\right\rfloor\right\} \\
&= 1 + \max_{\Delta t>0}\left\{\underbrace{\left\lceil\frac{\Delta t}{T_1}\right\rceil - \left\lfloor\frac{\Delta t}{T_1}\right\rfloor}_{=\begin{cases}0 & \text{if } \frac{\Delta t}{T_1}\in\mathbb{N}\\ 1 & \text{otherwise}\end{cases}}\right\} \\
&= 1+1=2
\end{aligned} \tag{6.11}$$

The minimum backlog can be calculated in a similar fashion, only that we are looking for infima [minima] instead of suprema [maxima]:

$$\begin{aligned}
\mathit{backlog}^{-}_{\mathcal{EAF}} &= \inf_{\Delta t>0}\left\{\eta^{-}_{\mathcal{EAF},\text{av}}(\Delta t) - \eta^{+}_{\mathcal{EAF},\text{out}}(\Delta t)\right\} \\
&= \min_{\Delta t>0}\left\{\left\lfloor\frac{\Delta t}{T_1}\right\rfloor + 1 - \left\lceil\frac{\Delta t}{T_1}\right\rceil\right\} \\
&= 1 - \max_{\Delta t>0}\left\{\left\lceil\frac{\Delta t}{T_1}\right\rceil - \left\lfloor\frac{\Delta t}{T_1}\right\rfloor\right\} \\
&= 1-1=0
\end{aligned} \tag{6.12}$$

Both values $\mathit{backlog}^{-}_{\mathcal{EAF}} = 0$ and $\mathit{backlog}^{+}_{\mathcal{EAF}} = 2$ are unsurprising. The minimum backlog corresponds to the minimum buffering requirements and can be found in case 2. These should be zero, otherwise we would have over-

or under-dimensioned the system, mainly through a inappropriate number of pre-loaded events in the buffer. The maximum buffer size of two is necessary in case 1.

#### 6.2.5.2 Buffering Delay

We mentioned that, for arbitrary curves and functions, the delay can not be calculated in a straight-forward way (see Equation 6.10). The actual calculations depend heavily on the $\eta$ functions. Fortunately, the use of standard event models provides us with well-formed curves, and we can follow a constructive approach. We create the inverse functions of $\eta$, and the sought-after delay corresponds to the vertical distance between the two inverse curves. The vertical distance can be calculated straight-forwardly, as Equation 6.9 shows.

Let two functions $\Gamma^-(n)$ and $\Gamma^+(n)$ determine the minimum and maximum time delay until the $n$th next event arrival for any possible point in time $t$. These functions represent the inverse of the $\eta$ functions, such that

$$\eta^+(\Gamma^-(n)) = n, \quad \text{and} \quad \eta^-(\Gamma^+(n)) = n.$$

With these functions, Equation 6.10 turns into:

$$\mathit{delay}_{\mathcal{EAF}} = \sup_{n>0} \{\Gamma_{\mathcal{EAF},\text{out}}(n) - \Gamma_{\mathcal{EAF},\text{in}}(n)\} \tag{6.13}$$

The actual $\Gamma$ functions can be derived as follows. $\Gamma^-(n)$, as the inverse of $\eta^+(\Delta t)$ determines the shortest possible delay $\Delta t$ until the arrival of the $n$th next event, measured from any possible point in time $t$. For the minimum delay, we have to assume immediate event arrival at $\Delta t = 0$, and the subsequent events arrive as soon as possible. We start with the shortest delay until the 1st event arrival, which apparently is zero:

$$\Gamma^-(1) = 0.$$

For all other $n > 1$, we can use the $\delta^-$ function that we introduced in Chapter 3. This $\delta^-(n)$ function determines the minimum distance between $n$ successive events. This is identical with the minimum delay until the $n$th next event arrival. Again, we assume immediate arrival of the 1st next event. So, $\Gamma^-(n)$ is given by

$$\forall\, n > 0 : \Gamma^-(n) = \begin{cases} 0 & \text{if } n = 1 \\ \delta^-(n) & \text{if } n > 1 \end{cases} \tag{6.14}$$

The observation that the first event just defines the beginning of the observation interval $\Delta t$ helped in deriving the $\Gamma^-$ function. The opposite assumption lets us define $\Gamma^+(n)$. Again, we have to assume that an event arrival defines the beginning of the observation interval, but this time, we have to assume that this event has just been missed and is not actually included. It rather appears as the

0th event, and all other events arrive as late as possible. Hence, we are looking for the maximum distance between $n$ plus one (the 0th one) events:

$$\forall\, n > 0 : \Gamma^+(n) = \delta^+(n+1) \tag{6.15}$$

The minimum and maximum distance of the shaper's periodic output events are known from Equation 3.13 in Chapter 3. It remains to determine the corresponding functions for the available events in the shaper buffer. We have obtained the $\eta$ function of the $\mathcal{EAF}$ output by moving the curve up by one event. For the $\Gamma$, and subsequently the $\delta$ functions, this corresponds to moving the curves to the right by one event:

$$\forall\, n > 0 : \Gamma_{\mathcal{EAF},\text{av}}(n) = \begin{cases} 0 & \text{if } n = 1 \\ \Gamma_{\mathcal{EAF},\text{in}}(n-1) & \text{if } n > 1 \end{cases} \tag{6.16}$$

The first event ($n = 1$) is immediately available ($\Gamma = 0$), since it is pre-loaded into the buffer.

We can now determine the minimum and maximum delay of the buffer using Equations 6.13, 6.14, 6.15, 6.16, and the characteristic $\delta$ function given by Equation 3.13. We start with the maximum delay. Since the property "delay" does not meaningfully apply to the pre-loaded event, we consider the distance between the curves only for $n > 1$:

$$\begin{aligned}
\mathit{delay}^+_{\mathcal{EAF}} &= \sup_{n>1} \left\{ \Gamma^+_{\mathcal{EAF},\text{out}}(n) - \Gamma^-_{\mathcal{EAF},\text{av}}(n) \right\} \\
&= \max_{n>1} \left\{ \Gamma^+_{\mathcal{EAF},\text{out}}(n) - \Gamma^-_{\mathcal{EAF},\text{in}}(n-1) \right\} \\
&= \max \left( \Gamma^+_{\mathcal{EAF},\text{out}}(2) - 0, \max_{n>2} \left\{ \Gamma^+_{\mathcal{EAF},\text{out}}(n) - \Gamma^-_{\mathcal{EAF},\text{in}}(n-1) \right\} \right) \\
&= \max \left( \delta^+_{\mathcal{EAF},\text{out}}(3), \max_{n>2} \left\{ \delta^+_{\mathcal{EAF},\text{out}}(n+1) - \delta^-_{\mathcal{EAF},\text{in}}(n-1) \right\} \right) \\
&= \max \left( 2\,T_1, \max_{n>2} \{ nT_1 - (n-2)T_1 \} \right) \\
&= \max\left(2\,T_1, 2\,T_1\right) \\
&= 2\,T_1
\end{aligned} \tag{6.17}$$

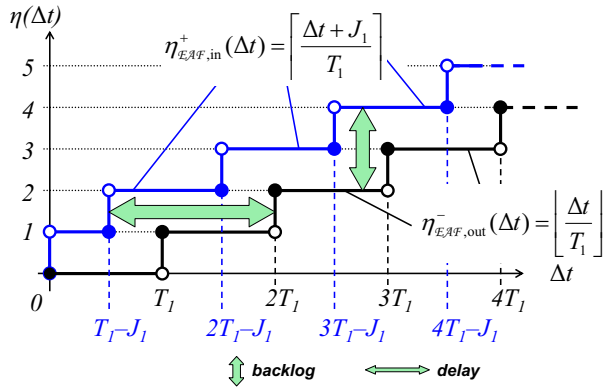



*Figure 6.10.* Synchronization of Jitter: Case 1 without Adaptation

We conclude this Section with the calculation of the minimum buffering delay:

$$\begin{aligned} delay^{-}_{\mathcal{EAF}} &= \inf_{n>1}\left\{\Gamma^{-}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{+}_{\mathcal{EAF},\text{av}}(n)\right\} \\ &= \min_{n>1}\left\{\Gamma^{-}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{+}_{\mathcal{EAF},\text{in}}(n-1)\right\} \\ &= \min_{n>1}\left\{\delta^{-}_{\mathcal{EAF},\text{out}}(n) - \delta^{+}_{\mathcal{EAF},\text{in}}(n)\right\} \\ &= \min_{n>1}\{(n-1)T_1 - (n-1)T_1\} \\ &= 0 \end{aligned} \tag{6.18}$$

We can find both minimum and maximum delay in the Figures 6.9 and 6.8.

## 6.3 Periodic Shaping of Jitter and Burst

Above, we have demonstrated and formally captured the influence of periodic shaping applied to strictly periodic streams. If the shapers input stream exhibits jitter (or burst), the situation is more complex. However, we can use the same ideas to determine the backlog and the delay in both known corner cases.

### 6.3.1 Small Jitters

We start with considering jitters less than periods. Figure 6.10 shows the shapers input and output curves of case 1, where the input arrives early and is

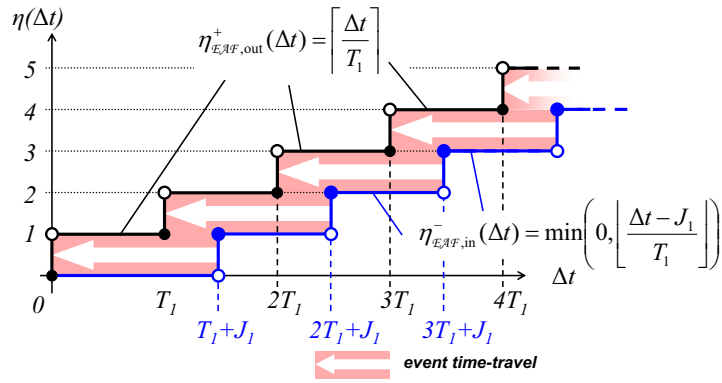



*Figure 6.11.* Synchronization of Jitter: Case 2 without Adaptation

outputted late. We see that the maximum buffering delay is $2T_1 - (T_1 - J_1) = T_1 + J_1$ and the required buffer size is two.

Case 2 with late input and early output is shown in Figure 6.11. Again, we see that events would need to time-travel. In contrast to the case of strictly periodic shaper input, the elimination of jitter requires two events, since the 2nd event is outputted even before the first one arrives at the shapers input. Therefore, the event availability functions $\eta_{\mathcal{EAF},\text{av}}$ is:

$$\eta_{\mathcal{EAF},\text{av}}(\Delta t) = \eta_{\mathcal{EAF},\text{in}}(\Delta t) + 2 = \eta_{1,\text{out}}(\Delta t) + 2$$

#### 6.3.1.1 Buffering Backlog

We can now calculate the maximum and minimum backlog.

$$\begin{aligned}
\mathit{backlog}^{+}_{\mathcal{EAF}} &= \sup_{\Delta t>0} \left\{ \eta^{+}_{\mathcal{EAF},\text{av}}(\Delta t) - \eta^{-}_{\mathcal{EAF},\text{out}}(\Delta t) \right\} \\
&= \max_{\Delta t>0} \left\{ \eta^{+}_{\mathcal{EAF},\text{in}}(\Delta t) + 2 - \eta^{-}_{\mathcal{EAF},\text{out}}(\Delta t) \right\} \\
&= \max_{\Delta t>0} \left\{ \left\lceil \frac{\Delta t + J_1}{T_1} \right\rceil + 2 - \left\lfloor \frac{\Delta t}{T_1} \right\rfloor \right\} \\
&= 2 + \max_{\Delta t>0} \left\{ \underbrace{\left\lceil \frac{\Delta t + J_1}{T_1} \right\rceil - \left\lfloor \frac{\Delta t}{T_1} \right\rfloor}_{= \begin{cases} 1 & \text{if } (\Delta t \bmod T_1) + J < T_1 \\ 2 & \text{otherwise} \end{cases}} \right\} \\
&= 2 + 2 = 4
\end{aligned} \tag{6.19}$$

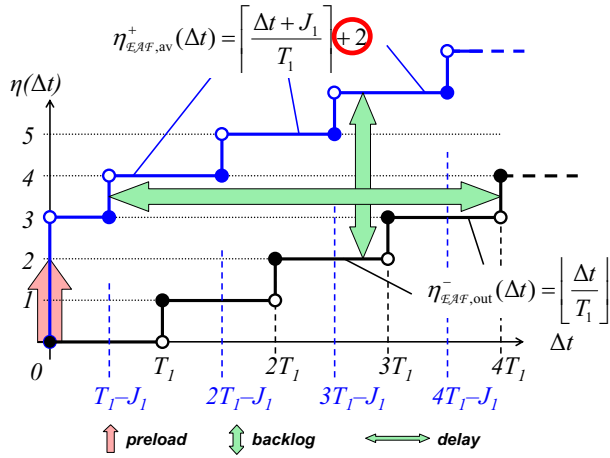



*Figure 6.12.* Synchronization of Jitter: Case 1 with Adaptation

$$
\begin{aligned}
\mathit{backlog}^-_{\mathcal{EAF}} &= \inf_{\Delta t>0} \left\{ \eta^-_{\mathcal{EAF},\mathrm{av}}(\Delta t) - \eta^+_{\mathcal{EAF},\mathrm{out}}(\Delta t) \right\} \\
&= \min_{\Delta t>0} \left\{ \eta^-_{\mathcal{EAF},\mathrm{in}}(\Delta t) + 2 - \eta^+_{\mathcal{EAF},\mathrm{out}}(\Delta t) \right\} \\
&= \min_{\Delta t>0} \left\{ \max\left(0, \left\lfloor \frac{\Delta t - J_1}{T_1} \right\rfloor\right) + 2 - \left\lceil \frac{\Delta t}{T_1} \right\rceil \right\}) \\
&= 2 - \max_{\Delta t>0} \left\{ \left\lceil \frac{\Delta t}{T_1} \right\rceil - \left\lfloor \frac{\Delta t - J_1}{T_1} \right\rfloor \right\} \\
&= 2 - 2 = 0
\end{aligned}
\tag{6.20}
$$

We see that the maximum backlog is four events. This is twice as much as in the case of purely periodic event production. The additional uncertainty caused by the jitter a) requires one more pre-load event, and b) can result in one more "early" event. The minimum backlog is –again– zero. Both properties can be seen in Figures 6.12 and 6.13.

#### 6.3.1.2 Buffering Delay

In order to determine the minimum and maximum buffering delay, we first need the $\Gamma$ function. When the $\eta$ functions are shifted up by two events, the $\Gamma$

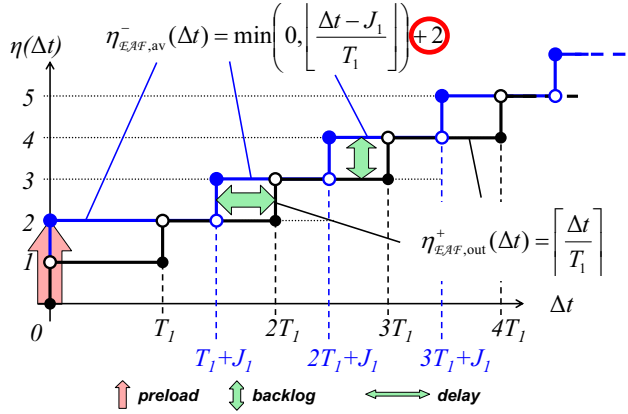



*Figure 6.13.* Synchronization of Jitter: Case 2 with Adaptation

needs to be shifted right by two events:

$$\forall\, n > 0 : \Gamma_{\mathcal{EAF},\mathrm{out}}(n) = \begin{cases} 0 & \text{if } n \leq 2 \\ \Gamma_{1,\mathrm{out}}(n-2) & \text{if } n > 2 \end{cases} \tag{6.21}$$

Now, we can determine the delay following Equation 6.13. We need not look at $n \leq 2$:

$$\begin{aligned}
\mathit{delay}^+_{\mathcal{EAF}} &= \sup_{n>2} \left\{ \Gamma^+_{\mathcal{EAF},\mathrm{out}}(n) - \Gamma^-_{\mathcal{EAF},\mathrm{av}}(n) \right\} \\
&= \max_{n>2} \left\{ \Gamma^+_{\mathcal{EAF},\mathrm{out}}(n) - \Gamma^-_{\mathcal{EAF},\mathrm{in}}(n-2) \right\} \\
&= \max\left( \Gamma^+_{\mathcal{EAF},\mathrm{out}}(3), \max_{n>3} \left\{ \Gamma^+_{\mathcal{EAF},\mathrm{out}}(n) - \Gamma^-_{\mathcal{EAF},\mathrm{in}}(n-2) \right\} \right) \\
&= \max\left( \delta^+_{\mathcal{EAF},\mathrm{out}}(4), \max_{n>3} \left\{ \delta^+_{\mathcal{EAF},\mathrm{out}}(n+1) - \delta^-_{\mathcal{EAF},\mathrm{in}}(n-2) \right\} \right) \\
&= \max\left( \underbrace{3\,T_1}_{\text{first event}}, \max_{n>3} \left\{ nT_1 + J_1 - \max\left(0, (n-3)T_1\right) \right\} \right) \\
&= \max\left( \underbrace{3\,T_1}_{\text{first event}}, \max_{n>3} \left\{ \underbrace{nT_1 + J_1 - (n-3)T_1}_{=3\,T_1+J_1} \right\} \right) \\
&= 3\,T_1 + J_1
\end{aligned} \tag{6.22}$$

We see that the first event possibly experiences a shorter delay than all later ones. This can also be seen in Figure 6.12. The minimum delay is:

$$\begin{aligned} \mathit{delay}^{-}_{\mathcal{EAF}} &= \inf_{n>2}\left\{\Gamma^{-}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{+}_{\mathcal{EAF},\text{av}}(n)\right\} \\ &= \min_{n>2}\left\{\Gamma^{-}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{+}_{\mathcal{EAF},\text{in}}(n-2)\right\} \\ &= \min_{n>2}\left\{\delta^{-}_{\mathcal{EAF},\text{out}}(n) - \delta^{+}_{\mathcal{EAF},\text{in}}(n-1)\right\} \\ &= \min_{n>2}\left\{nT_1 - ((n-1)T_1 + J_1)\right\} \\ &= T_1 - J_1 \end{aligned} \tag{6.23}$$

Interestingly, the minimum delay is not zero, as it was for periodic production in Section 6.2. The pre-load of two events is necessary to cover the worst-case jitter which can be any value less than the period $T_1$. Smaller jitters result is additional delays. Equation 6.23 shows that this additional delay increases with decreasing jitters.

### 6.3.2 Large Jitters and Bursts

Intuitively, larger jitters increase the necessary buffer pre-load, along with the overall buffer size and the delay. The actual numbers depend on the size of the jitter relative to the period. We assume a number of $k$ pre-loaded events in the shaper buffer. The event availability function is:

$$\eta_{\mathcal{EAF},\text{av}}(\Delta t) = \eta_{\mathcal{EAF},\text{in}}(\Delta t) + k \tag{6.24}$$

In order to avoid buffer underrun, we have to choose $k$ such that the backlog is at least zero. We also want to be optimal, i. e. we are looking for the smallest integer $k$ that fulfills the following inequation:

$$\begin{aligned} & & \mathit{backlog}^{-}_{\mathcal{EAF}} &\overset{!}{\geq} 0 \\ \Rightarrow\ & \eta^{-}_{\mathcal{EAF},\text{av}}(\Delta t) - \eta^{+}_{\mathcal{EAF},\text{out}}(\Delta t) &&\geq 0 \\ \Leftrightarrow\ & \eta^{-}_{\mathcal{EAF},\text{in}}(\Delta t) + k &&\geq \eta^{+}_{\mathcal{EAF},\text{out}}(\Delta t) \\ \Leftrightarrow\ & k &&\geq \eta^{+}_{\mathcal{EAF},\text{out}}(\Delta t) - \eta^{-}_{\mathcal{EAF},\text{in}}(\Delta t) \\ \Leftrightarrow\ & k &&\geq \left\lceil \tfrac{\Delta t}{T_1} \right\rceil - \max\left(0, \left\lfloor \tfrac{\Delta t - J_1}{T_1} \right\rfloor\right) \\ \Leftrightarrow\ & k &&\geq \left\lceil \tfrac{T_1 + J_1}{T_1} \right\rceil \\ \Rightarrow\ & k_{\min} &&= 1 + \left\lceil \tfrac{J_1}{T_1} \right\rceil \end{aligned}$$

This is the general formula to determine how many pre-loaded events are required. The formula also applies to small or zero jitter. In case of jitters less than periods, $k$ turns into 2, and for zero jitter $k = 1$, just as in the previous sections.

#### 6.3.2.1 Buffering Backlog

With a pre-load of $k_{\min}$ events, the minimum backlog is apparently zero, any larger value leads to undesired inefficiencies. With a known $k$, we can now actually calculate the maximum backlog:

$$
\begin{aligned}
backlog^{+}_{\mathcal{EAF}} &= \sup_{\Delta t>0}\left\{\eta^{+}_{\mathcal{EAF},\text{av}}(\Delta t) - \eta^{-}_{\mathcal{EAF},\text{out}}(\Delta t)\right\} \\
&= \max_{\Delta t>0}\left\{\eta^{+}_{\mathcal{EAF},\in}(\Delta t) + k_{\min} - \eta^{-}_{\mathcal{EAF},\text{out}}(\Delta t)\right\} \\
&= \max_{\Delta t>0}\left\{ \underbrace{\min\left(\left\lceil\frac{\Delta t + J_1}{T_1}\right\rceil, \left\lceil\frac{\Delta t}{d_1}\right\rceil\right)}_{= \left\lceil\frac{\Delta t + J_1}{T_1}\right\rceil \text{ for } \Delta t \geq \frac{J_1\, d_1}{T_1 - d_1},} + k_{\min} - \left\lfloor\frac{\Delta t}{T_1}\right\rfloor \right\} \\
&\qquad\qquad \text{see Equation 4.15} \\
&= \max_{\Delta t>0}\left\{\left\lceil\frac{\Delta t + J_1}{T_1}\right\rceil + k_{\min} - \left\lfloor\frac{\Delta t}{T_1}\right\rfloor\right\} \\
&= k_{\min} + \max_{\Delta t>0}\left\{ \underbrace{\left\lceil\frac{\Delta t + J_1}{T_1}\right\rceil - \left\lfloor\frac{\Delta t}{T_1}\right\rfloor}_{= \begin{cases} \lceil\frac{J_1}{T_1}\rceil & \text{if } (\Delta t \bmod T_1) \\ & \quad +(J_1 \bmod T_1) < T_1 \\ \lceil\frac{J_1}{T_1}\rceil + 1 & \text{otherwise} \end{cases}} \right\} \\
&= k_{\min} + \left\lceil\frac{J_1}{T_1}\right\rceil + 1 \\
&= k_{\min} + k_{\min} = 2k_{\min}
\end{aligned}
\tag{6.25}
$$

We can see that the burst-specific parameter $d_1$, capturing the minimum event distance during a burst, has no influence on the backlog and, as we will see in the next paragraph, on the delay.

#### 6.3.2.2 Buffering Delay

The maximum delay is:

$$
\begin{aligned}
\mathit{delay}^{+}_{\mathcal{EAF}} &= \sup_{n>k}\left\{\Gamma^{+}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{-}_{\mathcal{EAF},\text{av}}(n)\right\} \\
&= \max_{n>k}\left\{\Gamma^{+}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{-}_{\mathcal{EAF},\text{in}}(n-k)\right\} \\
&= \max\left(\Gamma^{+}_{\mathcal{EAF},\text{out}}(k+1),\ \max_{n>k+1}\left\{\Gamma^{+}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{-}_{\mathcal{EAF},\text{in}}(n-k)\right\}\right) \\
&= \max\left(\delta^{+}_{\mathcal{EAF},\text{out}}(k+2),\ \max_{n>k+1}\left\{\delta^{+}_{\mathcal{EAF},\text{out}}(n+1) - \delta^{-}_{\mathcal{EAF},\text{in}}(n-k)\right\}\right) \\
&= \max\left((k+1)T_1,\ \max_{n>k+1}\left\{\underbrace{nT_1 - \big((n-k-1)T_1 - J_1\big)}_{=(k+1)\,T_1+J_1}\right\}\right) \\
&= (k+1)\,T_1 + J_1
\end{aligned}
\tag{6.26}
$$

Finally, the minimum delay is given by:

$$
\begin{aligned}
\mathit{delay}^{-}_{\mathcal{EAF}} &= \inf_{n>k}\left\{\Gamma^{-}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{+}_{\mathcal{EAF},\text{av}}(n)\right\} \\
&= \min_{n>k}\left\{\Gamma^{-}_{\mathcal{EAF},\text{out}}(n) - \Gamma^{+}_{\mathcal{EAF},\text{in}}(n-k)\right\} \\
&= \min_{n>k}\left\{\delta^{-}_{\mathcal{EAF},\text{out}}(n) - \delta^{+}_{\mathcal{EAF},\text{in}}(n-k+1)\right\} \\
&= \min_{n>k}\left\{(n-1)T_1 - (((n-k+1)-1)\,T_1 + J_1)\right\} \\
&= (k-1)T_1 - J_1
\end{aligned}
\tag{6.27}
$$

### 6.3.3 System-Level Influence of Jitter

An interesting observation is that the minimum distance $d_1$ that reduces transient event frequencies during a burst does not affect periodic shaping at all. Figure 6.12 illustrates that the maximum delay and backlog are observed in the region where the jitter characteristics dominate the stream behavior. We conclude the section about jitter and burst with an analysis of the relationship between the jitter and the shaper properties.

When we use the optimal, i. e. the smallest valid buffer pre-load of $k_{\min}$, the minimum delay is bounded by the period $T_1$:

$$
\begin{aligned}
\mathit{delay}^{-}_{\mathcal{EAF}} &= (k_{\min}-1)\,T_1 - J_1 \\
&= \left\lceil \frac{J_1}{T_1} \right\rceil T_1 - J_1 \\
\Rightarrow \qquad 0 &\le \mathit{delay}^{-}_{\mathcal{EAF}} < T_1
\end{aligned}
\tag{6.28}
$$

For the maximum delay, we obtain:

$$\begin{aligned} \mathit{delay}^{+}_{\mathcal{EAF}} &= (k_{\min}+1)\,T_1 + J_1 \\ &= \left(\left\lceil \frac{J_1}{T_1} \right\rceil + 2\right) T_1 + J_1 \\ &= \left\lceil \frac{J_1}{T_1} \right\rceil T_1 + J_1 + 2\,T_1 \\ \Rightarrow \qquad 2\,J_1 + 2\,T_1 &\leq \mathit{delay}^{+}_{\mathcal{EAF}} < 2\,J_1 + 3\,T_1 \end{aligned} \tag{6.29}$$

We can see that the maximum delay grows twice as fast as the jitter. This shows the enormous influence of jitter on the overall system timing. The same applies to the backlog:

$$\begin{aligned} \mathit{backlog}^{+}_{\mathcal{EAF}} &= k_{\min} + k_{\min} = 2k_{\min} \\ &= 2\left\lceil \frac{J_1}{T_1} \right\rceil \\ &\geq 2\,\frac{J_1}{T_1} \end{aligned} \tag{6.30}$$

### 6.3.4 Optimizations

The concepts of periodic shaping introduced so far conservatively assume a) a fully independent shaper timer, b) strict FIFO communication semantics, and c) a known fixed buffer pre-load. We have mentioned that this is not an unusual requirement. Signal processing applications constrain the buffer pre-load in order to implement a specific functionality [68]. These constraints enforce a conservative design as explained above. In other situations, the communication between the two tasks might be less constrained, e. g. if there is a pre-load or not does not affect the application. We can take control over the individual situations and easily eliminate the mentioned inefficiencies, such as unused buffer or large delays. Again, we start with the simple case of a purely periodic sender $\mathcal{T}_1$. Recall the buffering scenarios depicted in Figures 6.3 and 6.4. The former starts with an empty buffer while the later starts with one pre-loaded event. A shaper task can easily distinguish both situations from one another at system run-time and dynamically choose the appropriate buffer set-up. The natural solution is to start with a *dummy event* in the buffer which is skipped if unnecessary. Such a situation is depicted in Figure 6.14. Essentially the first arriving event allows prediction of the future timing and *the optimal buffer set-up* to be chosen. The corresponding input and output event curves are shown in Figure 6.15. We see that a pre-load does not need to be considered for corner-case analysis. In effect, the buffer size can be reduced from two to one event. And the delay is also reduced from $2T$ to $1T$. Moreover, we can implement the

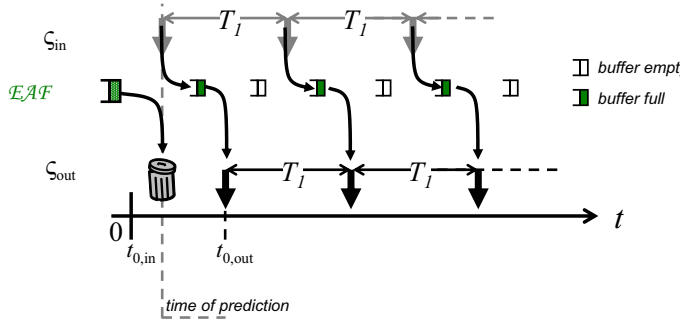



*Figure 6.14.* Overwriting an Unnecessarily Pre-Loaded Event

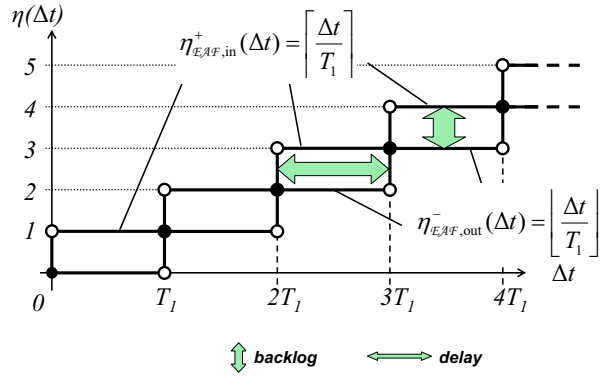



*Figure 6.15.* Event Curves with Optimized Buffer Set-Up

shaper using a simple register instead of a FIFO, since a register has exactly the right *destructive write* semantics that we need here.

In the case of task $\mathcal{T}_1$ producing events with jitter, the situation is slightly different. Depending on the actual jitter-to-period ratio, the corner cases differ by more than one required pre-load (see the linear dependency of $k_{\min}$ in Section 6.3), which results in more than two distinguishable situations, one for each possible buffer pre-load level. And secondly, a jitter adds uncertainty to the actual event arrival timing. This generally complicates the dynamic prediction of the actual situation. But we can use a modified FIFO buffer that is *initially empty*, and outputs *dummy events on underrun*. While we have used

an intermediate available event function (Equation 6.24) for the conservative calculations, the newly automatic *dynamic pre-load prediction* allows application of the backlog and delay calculations to the actual input and output event stream:

$$
\begin{aligned}
\mathit{backlog}^{+}_{\mathcal{EAF}} &= \sup_{\Delta t>0}\left\{\eta^{+}_{\mathcal{EAF},\text{in}}(\Delta t)-\eta^{-}_{\mathcal{EAF},\text{out}}(\Delta t)\right\}\\
&= \sup_{\Delta t>0}\left\{\left\lceil\frac{\Delta t+J_1}{T_1}\right\rceil-\left\lfloor\frac{\Delta t}{T_1}\right\rfloor\right\}\\
&= \sup_{\Delta t>0}\left\{\underbrace{\left\lceil\frac{\Delta t+J_1}{T_1}\right\rceil-\left\lfloor\frac{\Delta t}{T_1}\right\rfloor}\right\}\\
&\qquad =\begin{cases}\lceil\frac{J_1}{T_1}\rceil & \text{if } (\Delta t \bmod T_1)+\\ & \quad (J_1 \bmod T_1)<T_1\\ \lceil\frac{J_1}{T_1}\rceil+1 & \text{otherwise}\end{cases}\\
&= \left\lceil\frac{J_1}{T_1}\right\rceil+1=k_{\min}
\end{aligned}
\tag{6.31}
$$

We see that the required buffer size (the backlog) equals the required number of pre-loaded events $k_{\min}$ (see Equation 6.25). This is possible, since pre-loaded events are being substituted as new events arrive at the shapers input, preventing the buffer from overflowing.

The maximum delay is:

$$
\begin{aligned}
\mathit{delay}^{+}_{\mathcal{EAF}} &= \sup_{n>0}\left\{\Gamma^{+}_{\mathcal{EAF},\text{out}}(n)-\Gamma^{-}_{\mathcal{EAF},\text{in}}(n)\right\}\\
&= \max\left(\Gamma^{+}_{\mathcal{EAF},\text{out}}(1),\sup_{n>1}\left\{\Gamma^{+}_{\mathcal{EAF},\text{out}}(n)-\Gamma^{-}_{\mathcal{EAF},\text{in}}(n)\right\}\right)\\
&= \max\left(\delta^{+}_{\mathcal{EAF},\text{out}}(2),\sup_{n>1}\left\{\delta^{+}_{\mathcal{EAF},\text{out}}(n+1)-\delta^{-}_{\mathcal{EAF},\text{in}}(n)\right\}\right)\\
&= \max\left(\underbrace{T_1}_{\text{first event}},\sup_{n>1}\left\{\underbrace{nT_1-((n-1)T_1-J_1)}_{=T_1+J_1}\right\}\right)\\
&= T_1+J_1
\end{aligned}
\tag{6.32}
$$

We can now compare these results with those from the non-optimized FIFOs, given by Equations 6.30 and 6.29. We see that both the maximum delay and the maximum backlog still increase with increasing jitter $J_1$, but the values were reduced by a factor of two. The optimized shaper overwrites pre-loaded events, thereby circumventing the conservative buffer set-up, minimizing buffering requirements and reducing delays.

| task | $C$ | $T$ | $J$, $d$ | priority |
|---|---|---|---|---|
| $\mathcal{T}_1$ | 20 | 150 | - | 1 |
| $\mathcal{T}_2$ | 90 | 400 | 1100, 10 | 2 |
| $\mathcal{T}_3$ | 40 | 200 | - | 3 |

*Table 6.1.* Task Parameters of Example System

The mentioned optimizations, however, can temporarily disorder the events in the FIFO-communication, especially during the communication of the first events during system start-up. So, the optimizations can not be applied to systems which require the complete event order to be maintained, such as SDF.

### 6.3.5 Experiment

We will now illustrate the application of periodic shaping to a system of three tasks, $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$. Tasks $\mathcal{T}_1$ and $\mathcal{T}_3$ are activated strictly periodically with the periods $T_1$ and $T_3$, respectively. Task $\mathcal{T}_2$ is a reactive task, activated periodically (period $T_2$) but with a huge jitter $J_2$ that exceeds the period many times. The minimum event distance is 10. All tasks have different core execution times $C_1$, $C_2$, and $C_3$. Scheduling follows static priorities, task $\mathcal{T}_1$ has the highest priority, and task $\mathcal{T}_3$ the lowest. Table 6.1 summarizes the actual values. The average processor utilization is:

$$U = \sum_{i=1}^{3} \frac{C_i}{T_i} = \frac{20}{150} + \frac{90}{400} + \frac{40}{200} = 55.83\%$$

The *busy period* [69, 121] of the worst-case scheduling scenario is shown in Figure 6.16, generated by the SymTA/S tool [114], an in-house tool based on the key contributions of this thesis. In the depicted scheduling scenario, all tasks are activated simultaneously (critical instant, see Section 2.1.1) and, according to the upper-bound event arrival functions $\eta^+(\Delta t)$, re-arrive as early as possible. The small arrows mark activation times. We see the strictly periodic execution of the highest priority task $\mathcal{T}_1$ at $t_{\text{act},\mathcal{T}_1,1} = 0$, $t_{\text{act},\mathcal{T}_1,2} = 150$, $t_{\text{act},\mathcal{T}_1,3} = 300$, and so on. It is not preempted, and has also a strictly periodic completion behavior, i. e. the output jitter is zero.

Task $\mathcal{T}_2$ is activated three times within a very short time period, at $t_{\text{act},\mathcal{T}_2,1} = 0$ and $t_{\text{act},\mathcal{T}_2,2} = 10$, and $t_{\text{act},\mathcal{T}_2,3} = 20$, the fourth activation arrives at $t_{\text{act},\mathcal{T}_2,4} = 100$. This burst activation is due to the large jitter. We see that the first activation has not yet completed when the fourth activation arrives. Hence, the task's input queue must be able to store four tokens or events. In other words, the *execution backlog*, which is determined as a side effect of the arbitrary deadline analysis [69, 121], is 4 (four). The four executions are subsequently completed at $t_{\text{comp},\mathcal{T}_2,1} = 110$, $t_{\text{comp},\mathcal{T}_2,2} = 220$, $t_{\text{comp},\mathcal{T}_2,3} = 330$, and $t_{\text{comp},\mathcal{T}_2,4} = 420$.

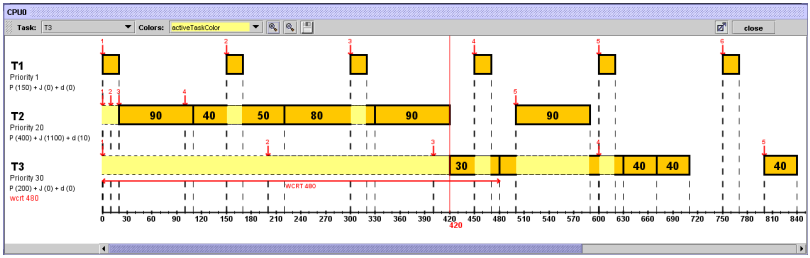



*Figure 6.16.* Scheduling Diagram of Un-shaped System

| task | execution backlog | response time | output jitter |
|---|---|---|---|
| $\mathcal{T}_1$ | 1 | 20 | - |
| $\mathcal{T}_2$ | 4 | 320 | 1330 |
| $\mathcal{T}_3$ | 3 | 480 | 440 |

*Table 6.2.* Experiment without Shaper

The maximum response time is observed for the fourth activation:

$$R_2^+ = t_{\text{comp},\mathcal{T}_2,4} - t_{\text{act},\mathcal{T}_2,4} = 420 - 100 = 320$$

Later activations arrive periodically with the period $T_2 = 300$, but need not be considered for worst-case response time analysis. The non-constant response times add further distortion to the input stream, resulting in an output jitter of 1330.

The lowest-priority task $\mathcal{T}_3$ is activated periodically at $t_{\text{act},\mathcal{T}_3,1} = 0$, $t_{\text{act},\mathcal{T}_3,2} = 200$, ..., but it is blocked until all activations of higher-level tasks have been completed at time $t = 420$. The first execution terminates at $t_{\text{comp},\mathcal{T}_3,1} = 480$. Meanwhile, a second and a third activation have arrived, which complete at $t_{\text{comp},\mathcal{T}_3,2} = 630$ and $t_{\text{comp},\mathcal{T}_3,2} = 670$. This burst execution requires an execution backlog buffer of 3 (three). The output jitter is 440, and the minimum distance is 40. The detailed results of this experiment are shown in Table 6.2.

We now insert an optimized periodic shaper at the input of task $\mathcal{T}_2$. We know that the required number of pre-loaded tokens equals the maximum backlog equals the required buffer size. This number $k_{\min}$ is given by Equation 6.25:

$$\mathit{backlog}^+_{\mathcal{EAF}} = \quad k_{\min} = 1 + \left\lceil \frac{J_2}{T_2} \right\rceil = 1 + \left\lceil \frac{1100}{400} \right\rceil = 4$$

Equation 6.32 yields the shaper's maximum buffering delay:

$$\mathit{delay}^+_{\mathcal{EAF}} = T_2 + J_2 = 400 + 1100 = 1500$$

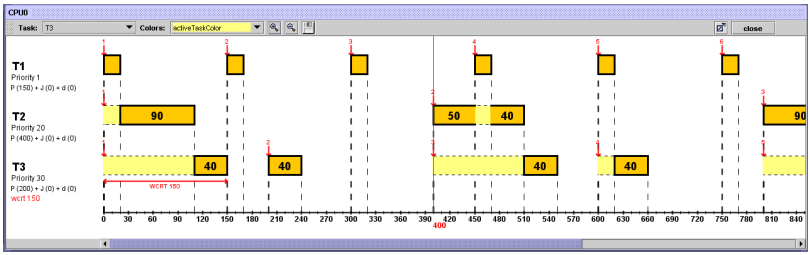



*Figure 6.17.* Scheduling Diagram of System with Periodic Shaper

| task | shaper backlog | exe. backlog | total buffer | buffer delay | task response | total delay | output jitter |
|---|---|---|---|---|---|---|---|
| $\mathcal{T}_1$ | - | 1 | 1 | - | 20 | 20 | - |
| $\mathcal{T}_2$ | 4 | 1 | **5** | 1500 | 110 | **1610** | **20** |
| $\mathcal{T}_3$ | - | 1 | **1** | - | 150 | **150** | **110** |

*Table 6.3.* Experiment with Periodic Shaper at Task $\mathcal{T}_2$'s Input

We need to store up to 4 tokens in the shaper buffer to synchronize the stream to its original period which results in a maximum buffering delay of 1500 time units. Now, the inputs of all tasks are strictly periodic resulting in a relatively deterministic schedule. The worst-case scheduling diagram is depicted in Figure 6.17. All tasks complete execution before their next activation, and the execution backlog is 1 (one). That means that we have fully eliminated the burst execution of tasks $\mathcal{T}_2$ and $\mathcal{T}_3$ and need to analyze the first activation of each task, only. We can directly read the worst-case response times from the diagram. The detailed results of this experiment are summarized in Table 6.3.

We see that the original system requires all together 8 buffer places to capture the execution backlog. The system with the periodic shaper, even though we have inserted an additional buffer into the system, requires only 7 buffer places, 4 to buffer the shaping backlog at task $\mathcal{T}_2$'s input, and one to buffer the execution backlog of each tasks. The elimination of the burst has even more positive effects on scheduling. The schedule is rather deterministic and the execution backlog of all tasks is one. In turn, the worst-case number of preemptions is heavily reduced, which tremendously reduces task response times: from 320 to 110 for task $\mathcal{T}_2$, and from 480 to 150 for task $\mathcal{T}_3$. Finally and maybe less obvious in this example, the output of a more deterministic task is also more deterministic. The output jitters have been heavily reduced, from 1330 to 20 for task $\mathcal{T}_2$, and from 440 to 110 for $\mathcal{T}_3$. This leads to more deterministic activation of a possibly connected task. In other words, the influence

of the inserted shaper is not limited to a single component. Rather, periodic shaping has a sustainable, positive effect on the entire system schedulability.

These improvements basically result from the equalization (or balancing) that shaping applies to jittery or bursty event streams. Essentially, the reduction of dynamic load peaks, such as bursts, reduces the timing uncertainty or non-determinism, in fact resulting in more static and regular system behavior. A comparison between Figures 6.16 and 6.17 illustrates the effects of more deterministic streams. The worst-case number of preemptions decreases, and task recurrence is fully eliminated.

Periodic shaping means synchronizing a jittery or bursty stream to its original period. The buffering delays that result from periodic shaping can be huge, and they increase with increasing jitter. In the above experiments, we see that shaping can add up to 1500 time units to the input events of task $\mathcal{T}_2$ events. The result is an *effective response time* of of task $\mathcal{T}_2$ of 1600 instead of 320 in the un-shaped system. Such an effect can cancel out other improvements, and periodic shaping becomes a trade-off among conflicting goals such as efficiency and predictability. However, we do not necessarily need to fully eliminate jitter and/or burst if we want to optimize the system, as the next section shows.

## 6.4 Sporadic Shaping

As a key impact on scheduling, periodic shaping produces *gaps in the schedule* which can now be exploited by other, lower-priority tasks to execute. This subsequently reduces the number of preemptions and finally leads to a more deterministic system. With periodic shaping, we pay for this improvement with long buffering delays. But to produce these gaps, we do not necessarily need to fully synchronize a jittery or bursty stream to its original period, if we can find other means to reduce the influence of bursts execution.

### 6.4.1 Transient Load Reduction

Where periodic shaping used a strictly periodic timer, sporadic shaping uses an equally simple *time-out mechanism*. Whenever an event is released at the shapers output, the time-out mechanism must make sure that no further events are released before a certain amount of time, the *minimum output distance*, has expired. Buffered events are released as soon as possible without violating the minimum output distance. The network theory refers to such shapers as *greedy shapers* [15, 14].

The time-out concept is easily applicable to the proposed set of event models, since the minimum output distance has a direct influence on the minimum inter-arrival time parameter of the burst models. When sporadic shaping with a time-out value of $d_{\mathcal{EAF},\text{out}}$ is applied to a periodic stream with jitter or burst

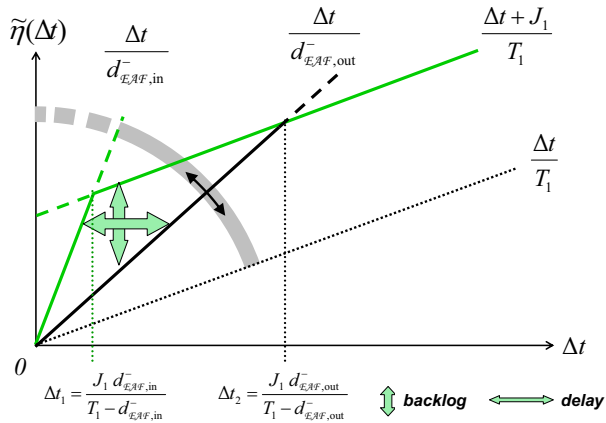



*Figure 6.18.* Influence of Sporadic Shaping on Upper-Bound Event Curves

($\mathcal{S}_{\mathcal{EAF},\text{in}}$), then the resulting output stream is given by:

$$\mathcal{S}_{\mathcal{EAF},\text{out}} = \mathcal{S}_{\mathsf{P+B}}(T_{\mathcal{EAF},\text{in}}, J_{\mathcal{EAF},\text{in}}, \max(d_{\mathcal{EAF},\text{out}}, \delta^-_{\mathcal{EAF},\text{in}}(2)) \qquad (6.33)$$

We see that the jitter is not affected by sporadic shaping but that a sporadic shaper imposes a minimum bound on the inter-arrival time of events. Figure 6.18 illustrates how the upper-bound event curve ($\eta^+(\Delta t)$) changes due to sporadic shaping. Bursts that in the original stream arrive with a minimum distance of $d_{\mathcal{EAF},\text{in}}$ are *slowed down* by increasing the allowed minimum distance to $d_{\mathcal{EAF},\text{out}}$. This slow-down corresponds to decreasing the initial slope of the $\eta^+$ curve. For large $\Delta t$, the function does not change.

The sporadic time-out value $d_{\mathcal{EAF},\text{out}}$ can be used to adjust the maximum transient frequencies of a stream. The gray region in Figure 6.18 illustrates the valid parameter range. The slow-down is clearly limited by the average slope, determined by the period $T_1$:

$$d_{\mathcal{EAF},\text{out}} \overset{!}{\leq} T_{\mathcal{EAF},\text{in}} = T_1 \qquad (6.34)$$

Larger values would apparently correspond to decreasing the average frequency and must be avoided. Values smaller than or equal to the original minimum distance do not change the timing of the stream, even though they are formally allowed.

With respect to the example from Section 6.3.5, we now replace the periodic shaper by a sporadic shaper with a time-out value of 200:

$$\mathcal{S}_{4,\text{in}} = \mathcal{S}_{\mathcal{EAF},\text{out}} = \mathcal{S}_{\mathsf{P+B}}(400, 1100, 200)$$

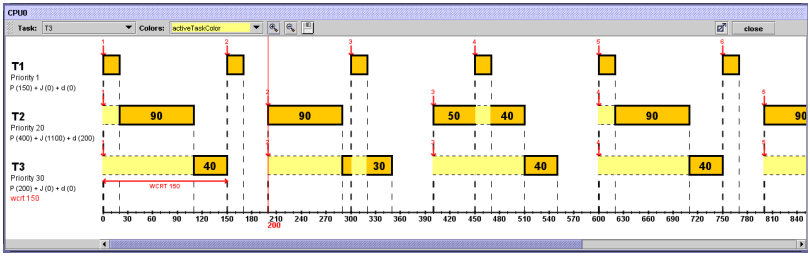



*Figure 6.19.* Scheduling Diagram of Sporadic Shaping with a Time-Out of 200

This means that input events of task $\mathcal{T}_2$ are delayed by the shaper such that they activate task $\mathcal{T}_2$ with a minimum distance of 200. Effectively, the second activation of task $\mathcal{T}_2$, which arrived at $t = 10$ without shaping (see Figure 6.16), is now delayed to arrive no earlier than at $t = 200$. We already know from Figure 6.17 that the first activation completes at time $t = 150$. An earliest possible re-activation of task $\mathcal{T}_2$ at $t = 200$ has obviously no influence on the worst-case scheduling scenario, as the diagram in Figure 6.19 shows. The execution backlog of all tasks is one.

### 6.4.2 Shaper Delay and Backlog

The network calculus ideas that were introduced in Section 6.2.5 can be applied to determine the shaper properties. But this time, the worst case buffering delay and backlog are found within the non-linear burst region, complicating the calculations. Instead of resolving the network calculus equations, we directly target at determining the curve distances graphically.

During a burst, all the incoming events have their minimum original distance, and all burst events except the first one have to be delayed. The second needs to be delayed only a little bit, the third one a bit more (more precisely, double as much as the second one), the third one three times as much as the first one, and so on. Finally, the last event of the burst is delayed for the longest time. It also leads to the highest buffer fill level. During *non-bursts*, no events need be buffered, hence the minimum delay and backlog are zero.

According to Equation 4.15, the burst period ends at

$$\Delta t_{\text{burst,in}} = \frac{J_1 \; d_{\mathcal{EAF},\text{in}}}{T_1 - d_{\mathcal{EAF},\text{in}}} \tag{6.35}$$

At that point, both the horizontal and vertical distances have their maximum values. But we have used the continuous event functions $\tilde{\eta}^+(\Delta t)$ to illustrate the problem in Figure 6.18. To determine the maximum backlog and delay for

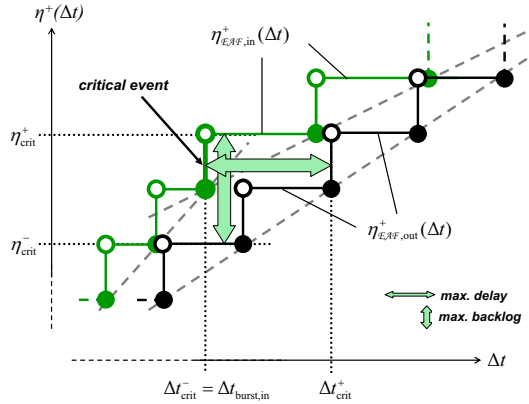



*Figure 6.20.* Single Critical Event in Sporadic Shaping

the discrete event functions, we need to identify the critical event that leads to worst-case buffering.

### 6.4.2.1 The Critical Event

If the last event in a burst arrives exactly at the theoretical end of the burst given by Equation 6.35, then there is only one critical event. This happens if the end of the burst $\Delta t_{\text{burst,in}}$ is an integer multiple of the minimum input distance $d_{\mathcal{EAF},\text{in}}$:

$$\frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} = \frac{\frac{J_1 \; d_{\mathcal{EAF},\text{in}}}{T_1 - d_{\mathcal{EAF},\text{in}}}}{d_{\mathcal{EAF},\text{in}}} = \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \stackrel{!}{\in} \mathbb{N} \tag{6.36}$$

Figure 6.20 illustrates the situation. The maximum backlog and delay are shown, bounded by $\Delta t^+_{\text{crit}} - \Delta t^-_{\text{crit}}$ and $\eta^+_{\text{crit}} - \eta^-_{\text{crit}}$, respectively. We start with $\eta^+_{\text{crit}}$:

$$\begin{aligned} \eta^+_{\text{crit}} &= \eta^+_{\mathcal{EAF},\text{in}}(\Delta t^-_{\text{crit}}) + 1 = \left\lceil \frac{\Delta t^-_{\text{crit}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil + 1 \\ &= \left\lceil \frac{\frac{J_1 \; d_{\mathcal{EAF},\text{in}}}{T_1 - d_{\mathcal{EAF},\text{in}}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil + 1 = \left\lceil \underbrace{\frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}}}_{\in \mathbb{N}} \right\rceil + 1 \end{aligned} \tag{6.37}$$

We have to add 1 to the value of the $\eta^+(\Delta t)$ function at that point because this function is left-continuous and has just not yet increased to $\eta^+_{\text{crit}}$ at $\Delta t^-_{\text{crit}}$.

Calculating $\eta^-_{\text{crit}}$ is straight-forward:

$$\eta^-_{\text{crit}} = \eta^+_{\mathcal{EAF},\text{out}}(\Delta t^-_{\text{crit}}) = \left\lceil \frac{\Delta t^-_{\text{crit}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil = \left\lceil \frac{\frac{J_1 \; d_{\mathcal{EAF},\text{in}}}{T_1 - d_{\mathcal{EAF},\text{in}}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil \tag{6.38}$$

We obtain as the maximum backlog:

$$\begin{aligned} \mathit{backlog}^+_{\mathcal{EAF}} &= \eta^+_{\text{crit}} - \eta^-_{\text{crit}} \\ &= 1 + \left\lceil \underbrace{\frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}}}_{\in \mathbb{N}} \right\rceil - \left\lceil \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \frac{d_{\mathcal{EAF},\text{in}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil \end{aligned} \tag{6.39}$$

To determine the maximum buffering delay, we calculate the delay between the $\eta^+_{\text{crit}}$th input and output event:

$$\begin{aligned} \mathit{delay}^+_{\mathcal{EAF}} &= \delta^-_{\mathcal{EAF},\text{out}}(\eta^+_{\text{crit}}) - \delta^-_{\mathcal{EAF},\text{in}}(\eta^+_{\text{crit}}) \\ &= (\eta^+_{\text{crit}} - 1) d_{\mathcal{EAF},\text{out}} - (\eta^+_{\text{crit}} - 1) d_{\mathcal{EAF},\text{in}} \\ &= (\eta^+_{\text{crit}} - 1)(d_{\mathcal{EAF},\text{out}} - d_{\mathcal{EAF},\text{in}}) \\ &= \left\lceil \frac{\Delta t_{\text{crit}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil (d_{\mathcal{EAF},\text{out}} - d_{\mathcal{EAF},\text{in}}) \\ &= \left\lceil \underbrace{\frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}}}_{\in \mathbb{N}} \right\rceil (d_{\mathcal{EAF},\text{out}} - d_{\mathcal{EAF},\text{in}}) \end{aligned} \tag{6.40}$$

Equation 6.36 ensures that some of the rounded terms are integers and do not essentially need rounding. We have not further simplified this because the same equations will be soon applied to other non-integer situations.

#### 6.4.2.2 Two Critical Event Candidates

If the last event in a burst does not exactly meet the end of the burst $\Delta t_{\text{burst,in}}$ given by Equation 6.35, then there are two candidates for the critical event, the last event *directly before* and the first event *directly after* $\Delta t_{\text{burst,in}}$. Obviously, one of these events experiences the maximum backlog and delay, respectively. We therefore determine these properties for both events. The two candidates are shown in Figure 6.21. We start by determining $\Delta t_{\text{crit},1}$ which is the largest integer multiple of the minimum input distance $d_{\mathcal{EAF},\text{in}}$ less than or equal to $\Delta t_{\text{burst,in}}$:

$$\Delta t_{\text{crit},1} = \left\lfloor \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rfloor d_{\mathcal{EAF},\text{in}} \tag{6.41}$$

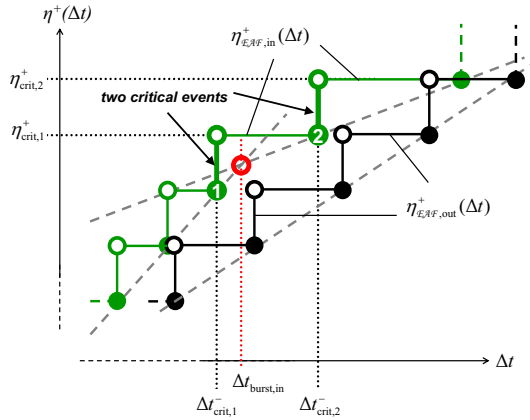



*Figure 6.21.* Two Critical Events in Sporadic Shaping

The backlog of this event can be calculated using Equations 6.37, 6.38, and 6.39:

$$
\begin{aligned}
\mathit{backlog}^+_{\mathcal{EAF},1} &= \eta^+_{\text{crit},1} - \eta^-_{\text{crit},1} \\
&= \eta^+_{\mathcal{EAF},\text{in}}(\Delta t_{\text{crit},1}) + 1 - \eta^+_{\mathcal{EAF},\text{out}}(\Delta t_{\text{crit},1}) \\
&= \left\lceil \frac{\Delta t_{\text{crit},1}}{d_{\mathcal{EAF},\text{in}}} \right\rceil + 1 - \left\lceil \frac{\Delta t_{\text{crit},1}}{d_{\mathcal{EAF},\text{out}}} \right\rceil \\
&= \left\lceil \frac{\left\lfloor \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rfloor d_{\mathcal{EAF},\text{in}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil + 1 - \left\lceil \frac{\left\lfloor \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rfloor d_{\mathcal{EAF},\text{in}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil \\
&= 1 + \left\lfloor \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rfloor - \left\lceil \left\lfloor \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rfloor \frac{d_{\mathcal{EAF},\text{in}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil \\
&= \underbrace{1 + \left\lfloor \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rfloor}_{\eta^+_{\text{crit},1}} - \underbrace{\left\lceil \left\lfloor \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rfloor \frac{d_{\mathcal{EAF},\text{in}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil}_{\eta^-_{\text{crit},1}}
\end{aligned}
\tag{6.42}
$$

The delay is obtained using Equation 6.40:

$$\begin{aligned} \mathit{delay}^{+}_{\mathcal{EAF},1} &= \delta^{-}_{\mathcal{EAF},\text{out}}(\eta^{+}_{\text{crit},1}) - \delta^{-}_{\mathcal{EAF},\text{in}}(\eta^{+}_{\text{crit},1}) \\ &= (\eta^{+}_{\text{crit},1} - 1)d_{\mathcal{EAF},\text{out}} - (\eta^{+}_{\text{crit},1} - 1)d_{\mathcal{EAF},\text{in}} \\ &= (\eta^{+}_{\text{crit},1} - 1)(d_{\mathcal{EAF},\text{out}} - d_{\mathcal{EAF},\text{in}}) \\ &= \left\lfloor \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rfloor (d_{\mathcal{EAF},\text{out}} - d_{\mathcal{EAF},\text{in}}) \end{aligned} \tag{6.43}$$

The arrival time of the second candidate can be determined similarly to the calculation in Equation 6.41, but this time we are looking for the event arrival time greater than or equal to $\Delta t_{\text{burst,in}}$; and because the event arrives *after* the end of the burst, we need to consider a periodic behavior with jitter rather than a sporadic behavior with a minimum distance:

$$\Delta t_{\text{crit},2} = \left\lceil \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil T_1 - J_1 \tag{6.44}$$

The backlog of this event is::

$$\begin{aligned} \mathit{backlog}^{+}_{\mathcal{EAF},2} &= \eta^{+}_{\text{crit},2} - \eta^{-}_{\text{crit},2} \\ &= \underbrace{\eta^{+}_{\mathcal{EAF},\text{in}}(\Delta t_{\text{crit},2})}_{\text{after end of input burst}} + 1 - \underbrace{\eta^{+}_{\mathcal{EAF},\text{out}}(\Delta t_{\text{crit},2})}_{\text{output still bursty}} \\ &= \left\lceil \frac{\Delta t_{\text{crit},2} + J_1}{T_1} \right\rceil + 1 - \left\lceil \frac{\Delta t_{\text{crit},2}}{d_{\mathcal{EAF},\text{out}}} \right\rceil \\ &= 1 + \left\lceil \frac{\left\lceil \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil T_1 - J_1 + J_1}{T_1} \right\rceil - \left\lceil \frac{\left\lceil \frac{\Delta t_{\text{burst,in}}}{d_{\mathcal{EAF},\text{in}}} \right\rceil T_1 - J_1}{d_{\mathcal{EAF},\text{out}}} \right\rceil \\ &= \underbrace{1 + \left\lceil \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rceil}_{\eta^{+}_{\text{crit},2}} - \underbrace{\left\lceil \frac{\left\lceil \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rceil T_1 - J_1}{d_{\mathcal{EAF},\text{out}}} \right\rceil}_{\eta^{-}_{\text{crit},2}} \end{aligned} \tag{6.45}$$

The delay is obtained using Equation 6.40:

$$\begin{aligned} \mathit{delay}^{+}_{\mathcal{EAF},2} &= \underbrace{\delta^{-}_{\mathcal{EAF},\text{out}}(\eta^{+}_{\text{crit},2})}_{\text{output still bursty}} - \underbrace{\delta^{-}_{\mathcal{EAF},\text{in}}(\eta^{+}_{\text{crit},2})}_{\text{after end of input burst}} \\ &= (\eta^{+}_{\text{crit},2} - 1)d_{\mathcal{EAF},\text{out}} - (\eta^{+}_{\text{crit},2} - 1)T_1 - J_1 \\ &= (\eta^{+}_{\text{crit},2} - 1)(d_{\mathcal{EAF},\text{out}} - T_1) + J_1 \\ &= \left\lceil \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rceil (d_{\mathcal{EAF},\text{out}} - T_1) + J_1 \end{aligned} \tag{6.46}$$

| task | shaper backlog | exe. backlog | total buffer | buffer delay | task response | total delay | output jitter |
|---|---|---|---|---|---|---|---|
| $\mathcal{T}_1$ | - | 1 | 1 | - | 20 | 20 | - |
| $\mathcal{T}_2$ | 3 | 1 | **4** | 500 | 110 | **610** | **1120** |
| $\mathcal{T}_3$ | - | 1 | 1 | - | 150 | 150 | 110 |

*Table 6.4.* Experiment with Sporadic Shaper with a Time-Out of 200

Finally, we choose the larger value of both candidates:

$$\begin{aligned} \mathit{backlog}^{+}_{\mathcal{EAF}} &= \max(\mathit{backlog}^{+}_{\mathcal{EAF},1}, \mathit{backlog}^{+}_{\mathcal{EAF},2}) \\ &= \max\left( \begin{array}{l} 1 + \left\lfloor \dfrac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rfloor - \left\lceil \left\lfloor \dfrac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rfloor \dfrac{d_{\mathcal{EAF},\text{in}}}{d_{\mathcal{EAF},\text{out}}} \right\rceil, \\ 1 + \left\lceil \dfrac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rceil - \left\lceil \dfrac{\left\lceil \frac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rceil T_1 - J_1}{d_{\mathcal{EAF},\text{out}}} \right\rceil \end{array} \right) \end{aligned} \tag{6.47}$$

$$\begin{aligned} \mathit{delay}^{+}_{\mathcal{EAF}} = \quad & \max( \quad \mathit{delay}^{+}_{\mathcal{EAF},1}, \mathit{delay}^{+}_{\mathcal{EAF},2}) \\ = \quad & \max\left( \begin{array}{l} \left\lfloor \dfrac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rfloor (d_{\mathcal{EAF},\text{out}} - d_{\mathcal{EAF},\text{in}}), \\ \left\lceil \dfrac{J_1}{T_1 - d_{\mathcal{EAF},\text{in}}} \right\rceil (d_{\mathcal{EAF},\text{out}} - T_1) + J_1 \end{array} \right) \end{aligned} \tag{6.48}$$

These are the general formulas for backlog and delay determination. In case of single critical event, both values $\Delta t_{\text{crit},1}$ and $\Delta t_{\text{crit},2}$ are identical and the general backlog calculations of Equation 6.47 can be reduced to those of Equation 6.39. The same applies to the delay calculations

### 6.4.3 Experiments

Now that we have derived a formal framework for sporadic shaping, we can determine the properties of the initial example. Detailed results are given in Table 6.4. Compared to the solution with the periodic shaper (see Table 6.3), the required shaper buffer can be reduced from 4 to 3, and the buffering delay has decreased from 1500 to 500, reducing the effective task $\mathcal{T}_2$ response time from 1610 to 610 time units. As a side-effect, the output jitter of task $\mathcal{T}_2$ could not be reduced.

In order to illustrate the application of sporadic shaping, we have varied the time-out value of the shaper. The next experiments start with the maximum

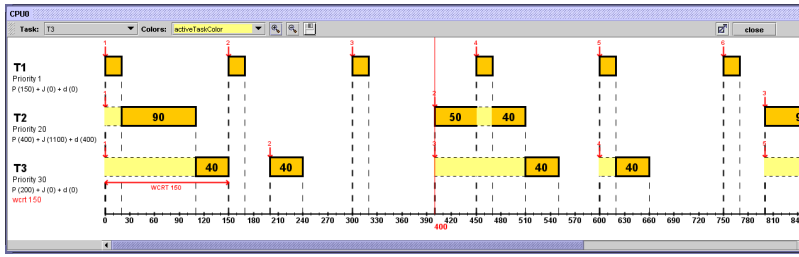



*Figure 6.22.* Scheduling Diagram of System Sporadic Shaper with a Time-Out of 400

| task | shaper backlog | exe. backlog | total buffer | buffer delay | task response | total delay | output jitter |
|---|---|---|---|---|---|---|---|
| $\mathcal{T}_1$ | - | 1 | 1 | - | 20 | 20 | - |
| $\mathcal{T}_2$ | 3 | 1 | 4 | 1100 | 110 | **1210** | 1120 |
| $\mathcal{T}_3$ | - | 1 | 1 | - | 150 | 150 | 110 |

*Table 6.5.* Experiment with Sporadic Shaper with a Time-Out of 400

possible time-out value, which is (see Equation 6.34) the original period of 400 time units. Then we gradually decrease the time-out value to 140, and finally 90 time-units. A detailed comparison shows the impact of shaping and the trade-offs that provide room for optimizations.

#### 6.4.3.1 400-Time Units Shaper

With a time-out of 400, we apply as much sporadic shaping as possible, larger time-out values are not possible. The worst-case scheduling diagram is shown in Figure 6.22, again being identical to those with periodic shaping, and also to sporadic shaping with a time-out of 200. The reason is –again– the large minimum distance that prevents task $\mathcal{T}_2$ from re-arriving during the so called critical scheduling instant. Table 6.5 shows the detailed results of the experiment. While the periodic shaper required a buffer of size 4, the maximum required buffer size of the sporadic shaper is only 3, just as in the case of a 200 time-unit time-out. The buffering delay is between the delays of the previous experiments.

#### 6.4.3.2 140-Time Units Shaper

A reduction of the time-out value below 150 time units results in more complex worst-case scheduling diagrams, since now, re-activations of task $\mathcal{T}_2$ need be considered during analysis. Figure 6.22 shows the worst-case scheduling diagram.

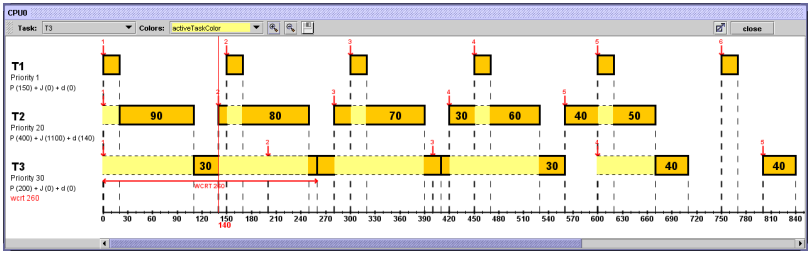



*Figure 6.23.* Scheduling Diagram of System Sporadic Shaper with a Time-Out of 140

| task | shaper backlog | exe. backlog | total buffer | buffer delay | task response | total delay | output jitter |
|---|---|---|---|---|---|---|---|
| $\mathcal{T}_1$ | - | 1 | 1 | - | 20 | 20 | - |
| $\mathcal{T}_2$ | 3 | 1 | 4 | 320 | 110 | **430** | 1120 |
| $\mathcal{T}_3$ | - | 2 | **2** | - | 260 | **260** | **220** |

*Table 6.6.* Experiment with Sporadic Shaper with a Time-Out of 140

Task $\mathcal{T}_2$ re-arrives at $t = 140$ and preempts $\mathcal{T}_3$ which, in turn, re-arrives at $t = 200$ before it has completed. Two more re-arrivals of $\mathcal{T}_2$ disturb the execution of $\mathcal{T}_3$ further. Finally, the critical scheduling instant is finished at $t = 560$. The figure shows that the response time of task $\mathcal{T}_2$ has not increased, since there is at most one preemption by $\mathcal{T}_1$. However, the lowest priority task $\mathcal{T}_3$ is now being preempted more often than with larger time-out values. Hence, its response time increases and even exceeds its own activation period, resulting in larger execution backlog. The detailed results are shown in Table 6.6.

We see that task $\mathcal{T}_3$ suffers from the decreasing time-out values, although the required shaper buffer remains constant compared to the previous experiment. However, the buffering (or shaping) delay has further decreased from 500 to 320.

#### 6.4.3.3 90-Time Units Shaper

In the final experiment, we reduce the time-out to 90. The scheduling diagram in Figure 6.24 illustrates that the resulting scheduling is already very complex, again. There is almost no difference compared to the un-shaped version of Figure 6.16. Only the total input backlog of task $\mathcal{T}_2$ is now partitioned into a shaping backlog of 2 and an execution backlog of 2, while the un-shaped version requires an execution backlog of 4. The detailed results are shown in Table 6.7.

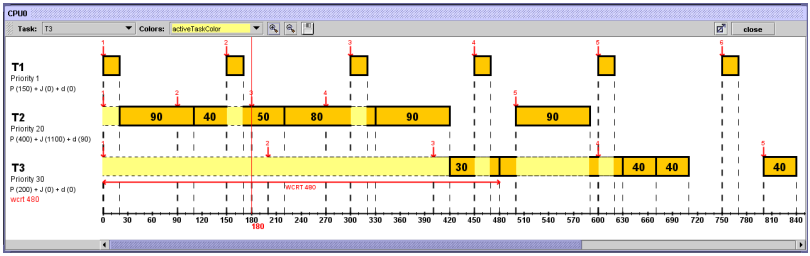



*Figure 6.24.* Scheduling Diagram of System Sporadic Shaper with a Time-Out of 90

| task | shaper backlog | exe. backlog | total buffer | buffer delay | task response | total delay | output jitter |
|---|---|---|---|---|---|---|---|
| $\mathcal{T}_1$ | - | 1 | 1 | - | 20 | 20 | - |
| $\mathcal{T}_2$ | 2 | 2 | 4 | 170 | 150 | **320** | 1160 |
| $\mathcal{T}_3$ | - | 3 | 3 | - | 480 | **480** | **440** |

*Table 6.7.* Experiment with Sporadic Shaper with a Time-Out of 90

#### 6.4.3.4 Experiment Summary

We have seen that very large time-out values result in the same worst-case scheduling as the periodic version with much less delay. In contrast, very small time-out values result in properties similar to the un-shaped version. In the middle range, there are interesting possibilities of optimizing the system by trading buffering requirements against delays against output jitters, and the individual task's delays against each other. Figure 6.25 allows a direct comparison of the worst-case scheduling scenarios of all experiments. The top scheduling diagram is taken from the experiment with a periodic shaper, the other diagrams illustrate the influence of decreasing time-out values from top to bottom. The bottom diagram shows the up-shaped version.

Table 6.8 provides the detailed results. The bold numbers represent optimal values for a specific property. We see that, in this experiment, each set-up is optimal with respect to a particular property. This is not necessarily the case, as the fourth row of Table 6.8 already indicates. The sporadic shaper with a time-out of 140 shares the optimal task $\mathcal{T}_2$ backlog with four other solutions and has non-optimal values for all other properties. However, it still represents a Pareto-optimal solution, since neither of the other solutions outperforms it in all properties. Usually, a higher number of time-out values results in a wider Pareto front of sub-optimal solutions, which can be traded against each other.

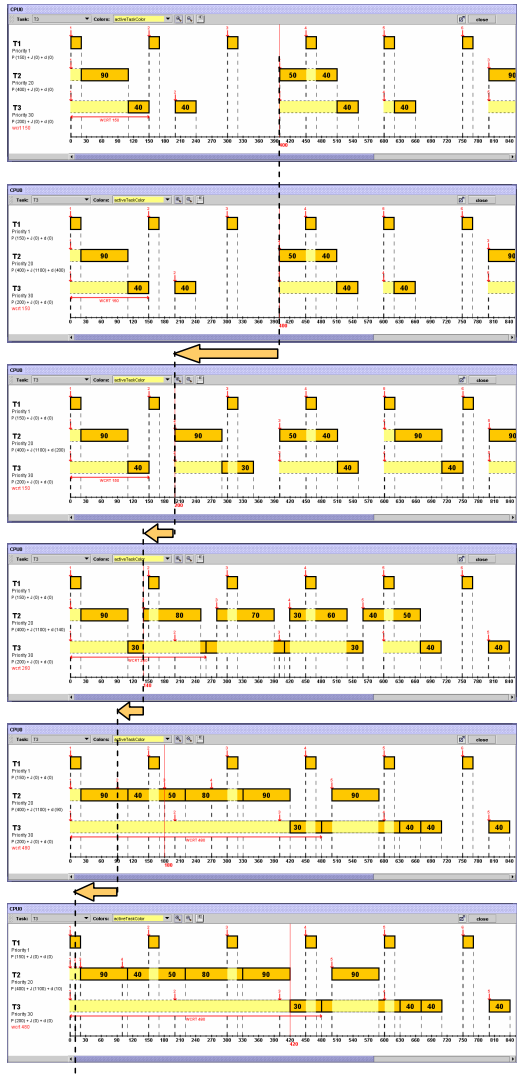



*Figure 6.25.* Scheduling Diagrams of All Experiments

| | | Shaping | | | | | |
|---|---|---|---|---|---|---|---|
| | | P | 400 | 200 | 140 | 90 | – |
| | | Accumulated Backlog (shaper + execution) | | | | | |
| Task | $\mathcal{T}_1$ | 1 | 1 | 1 | 1 | 1 | 1 |
| | $\mathcal{T}_2$ | 5 | **4** | **4** | **4** | **4** | **4** |
| | $\mathcal{T}_3$ | **1** | **1** | **1** | 2 | 3 | 3 |
| | Total | 7 | **6** | **6** | 7 | 8 | 8 |
| | | Accumulated Delay (shaper + response) | | | | | |
| Task | $\mathcal{T}_1$ | 20 | 20 | 20 | 20 | 20 | 20 |
| | $\mathcal{T}_2$ | 1610 | 1210 | 610 | 430 | **320** | **320** |
| | $\mathcal{T}_3$ | **150** | **150** | **150** | 260 | 480 | 480 |
| | | Output Jitter | | | | | |
| Task | $\mathcal{T}_1$ | - | - | - | - | - | - |
| | $\mathcal{T}_2$ | **20** | 1120 | 1120 | 1120 | 1160 | 1330 |
| | $\mathcal{T}_3$ | **110** | **110** | **110** | 220 | 440 | 440 |

*Table 6.8.* Overview of Delay and Backlog of All Experiments

### 6.4.4 Shaping Sporadic Streams

So far, we have applied shaping solely to *periodic* streams with jitter and burst. Periodic shaping eliminates a jitter by re-synchronizing the events to the original period, sporadic shaping reduces the strength of a burst by increasing the minimum inter-arrival time between events. It is quite obvious that periodic shaping cannot be applied to sporadic streams, at least not in the way presented in Section 6.2. But we can clearly apply sporadic shaping to sporadic streams. The same concepts that have been introduced also apply to sporadic shaping. The shaper's output stream will then be:

$$\mathcal{S}_{\mathcal{EAF},\text{out}} = \mathcal{S}_{\mathsf{S+B}}(T_{\mathcal{EAF},\text{in}}, J_{\mathcal{EAF},\text{in}}, \max(d_{\mathcal{EAF},\text{out}}, \delta^{-}_{\mathcal{EAF},\text{in}}(2)) \tag{6.49}$$

The maximum backlog and delay calculation from Equations 6.47 and 6.48 apply.

#### 6.4.4.1 Maximum Sporadic Shaping

There is, however, an interesting situation that one could call "sporadic synchronization". If we choose the maximum allowed time-out value $d_{\mathcal{EAF},\text{out}}$, which is equal to the original sporadic period $T_{\mathcal{EAF},\text{in}}$, then the resulting stream is:

$$\mathcal{S}_{\mathcal{EAF},\text{out}} = \mathcal{S}_{\mathsf{S+B}}(T_{\mathcal{EAF},\text{in}}, J_{\mathcal{EAF},\text{in}}, T_{\mathcal{EAF},\text{in}}) \tag{6.50}$$

We can now apply the additional event model interface from Section 5.6.3 to transform that stream into a strictly sporadic (S) stream:

$$\mathcal{S}_{\mathcal{EAF},\text{out}} = \mathcal{S}_{\mathsf{S}}(T_{\mathcal{EAF},\text{in}}) \tag{6.51}$$

In other words, under specific circumstances, we can apply a *lossless* event model transformation:

$$\begin{aligned} \mathcal{EMIF}_{\mathsf{S+B}\rightarrow\mathsf{S}} &: \{\mathcal{S}_{\mathsf{S+B}}(T,J,d) \in \mathcal{EM}_{\mathsf{S+B}} | d = T\} \mapsto \mathcal{EM}_{\mathsf{S}} \\ \mathcal{S}_{\text{target}} &= \mathcal{S}_{\mathsf{S}}(T_{\text{target}} = T_{\text{source}}) \end{aligned} \tag{6.52}$$

Together with a sporadic shaper, this interface allows sporadic streams to be *virtually re-synchronized* to their original sporadic period. The standard calculations apply to the dimensioning of the shaper buffer and to obtaining the shaping delay.

## 6.5 Automatic Shaping

So far, we have considered shaping as a means to *manually* optimize scheduling locally. There is, however, another very important application of shaping. We have seen in the previous chapter, that there exist situations, in which the requirement coverage test of Definition 5.4 fails, that compares the characteristic functions of a stream against the allowed range of the requirement. In these situations, no plain event model interface can be found. Shaping, however, changes the specific parameters of a stream, along with the characteristic functions. Apparently, a given input stream $\mathcal{S}_{i,\text{in}}$ can possibly be shaped such that the resulting stream meets a given input requirement $\mathcal{R}_{i,\text{in}}$ in cases where no plain $\mathcal{EMIF}$ is available.

We have shown in Section 6.4.3 that the actual impact of shaping varies heavily with the choice of the shaper. Depending on the shaping type, periodic or sporadic, and on the time-out value, task activation can be deliberately controlled and optimized by the designer. In other words, designers can exploit alternatives when shaping streams. However, for an *automatic procedure*, that we will introduce in Chapter 7, we need a clear, unambiguous rule to select a shaper. Therefore, we first define appropriate *goals of an automatic shaping*.

We can distinguish optimizations from necessities as follows. A task's input requirement must essentially be met, otherwise the task can not be analyzed. This is a necessity. When shaping cannot be avoided, we want to reduce the influence of shaping on the overall system to a minimum, i. e. we are looking for a shaper with the minimum backlog and delay. In other words, the input stream shall be shaped *as little as possible* but *as much as necessary*. Everything beyond these necessities can be subject to further user-controlled optimization, and is called *user-controlled shaping*.

**Definition 6.1.** ***(Automatic Shaper)***
*An Automatic Shaper $\mathcal{EAF}_{auto}$ is a function that takes as argument an event stream $\mathcal{S}_{\mathcal{EAF}_{auto},in}$ and a requirement $\mathcal{R}$, and returns a shaped stream $\mathcal{S}_{\mathcal{EAF}_{auto},out}$.*

*Let $\mathcal{Z}$ be the set of all possible event streams $\mathcal{S}$, and $\mathbb{P}(\mathcal{Z})$ be its power set. Then, $\mathcal{EAF}_{auto}$ is defined by:*

$$\begin{aligned} \mathcal{EAF}_{auto} &: \mathcal{Z} \times \mathbb{P}(\mathcal{Z}) \mapsto \mathcal{Z} \\ \mathcal{S}_{\mathcal{EAF}_{auto},out} &= \mathcal{EAF}_{auto}(\mathcal{S}_{\mathcal{EAF}_{auto},in}, \mathcal{R}) \end{aligned} \tag{6.53}$$

*If possible, the function shapes the provided event stream such, that*

*a) the shaper's output event stream is covered by the given requirement*

$$\mathcal{RC}(\mathcal{S}_{\mathcal{EAF}_{auto},out}, \mathcal{R}) = 1(true)\ , and \tag{6.54}$$

*b) the required amount of buffering and buffering delay is minimized. The corresponding parameters $backlog^{+}_{\mathcal{EAF}_{auto}}$, $backlog^{-}_{\mathcal{EAF}_{auto}}$, $delay^{+}_{\mathcal{EAF}_{auto}}$, and $delay^{-}_{\mathcal{EAF}_{auto}}$ are associated with the Automatic Shaper.*

*If there exists no shaper that fulfills the given requirement $\mathcal{R}$ according to Equation 6.54, the function returns the un-shaped source stream, and the four parameters have a value of zero (0).* ■

The experiments of Section 6.4.3 have shown us that sporadic shaping has a smaller influence than periodic shaping, and that smaller possible time-out values result in less buffering and less delay. Hence, *sporadic shapers are favored* over periodic shapers for automatic shaping. Furthermore, an automatic shaper must have the *minimum possible time-out value* that is just sufficient to meet the input requirement with respect to the allowed minimum event distance $\delta^{-}$. The minimum possible value is taken from all allowed streams ($\mathcal{R}_{i,\text{in}}$) that support the actual input period *and* the actual input jitter:

$$d_{\mathcal{EAF}_{\text{auto}},\text{out}} = \min_{\mathcal{S} \in \{\mathcal{S}_x \in \mathcal{R}_{i,\text{in}} | T_x = T_{i,\text{in}} \wedge J_x \geq J_{i,\text{in}}\}} (\delta^{-}_{\mathcal{S}}(2)) \leq \delta^{-}_{\mathcal{S}_{i,\text{in}}}(2) \tag{6.55}$$

Practically speaking, the minimum time-out value must be selected according to the maximum allowed transient input frequency, given that the period and the jitter are within the allowed range.

We have mentioned in Section 6.4.1 that sporadic shaping never reduces the jitter, it can only reduce transient event frequencies. In case the input jitter exceeds the maximum allowed jitter, then the set

$$\{\mathcal{S}_x \in \mathcal{R}_{i,\text{in}} | T_x = T_{i,\text{in}} \wedge J_x \geq J_{i,\text{in}}\}$$

is empty, and no appropriate $d_{\mathcal{EAF}_{\text{auto}},\text{out}}$ can be found using Equation 6.55. There is one exception, which we called *maximum sporadic shaping* in Section 6.4.4. This eliminates the jitter of a sporadic stream by choosing the maximum possible timeout value, which is the sporadic period $T_{\text{S}}$ of the stream.

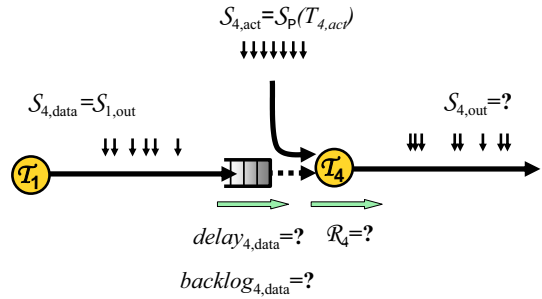



*Figure 6.26.* Periodic Polling Task and the Involved Event Streams

In case of the input stream is periodic with a too large jitter, a periodic shaper of Section 6.2 must be used. In the introductory example of this chapter, we have shown that the original period of the stream must be clearly compatible with the task input requirement (see Equation 6.4).

## 6.6 Polling

In Section 6.2, we have shown how periodic streams can be re-synchronized to their original period. An experiment has shown how such periodic shapers can result in a periodic execution of an otherwise very dynamic task, thereby substantially reducing peak loads and increasing the determinism.

Periodic task execution, however, does not necessarily always require strictly periodic inputs. *Polling* is a popular mechanism to also apply periodic execution to asynchronous inputs, including sporadic streams. The behavior of a polling task is influenced by *two event streams*. A strictly periodic one that activates the task, and another arbitrary one that influences the tasks functional behavior. Figure 6.26 shows a polling task $\mathcal{T}_4$ and the two streams, one activating event stream $\mathcal{S}_{4,\text{act}}$ and one data stream $\mathcal{S}_{4,\text{data}}$ coming from task $\mathcal{T}_1$.

In other words, the activation scheme does not reflect communication and vice versa. Rather, a polling task is not activated upon the availability of inputs but is time-driven. When activated, then it tests its inputs for available data, and the task's behavior might further depend on the success or failure of the test.

There are four parameters that must be determined, as illustrated in Figure 6.26. The buffer is needed to synchronize the data input to the periodic execution, and we need to determine the data buffering delay $delay^+_{4,\text{data}}$ and backlog $backlog^+_{4,\text{data}}$. Furthermore, the execution delay, i. e. the response time

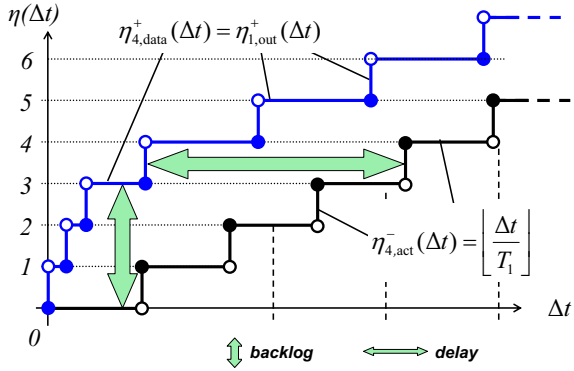



*Figure 6.27.* Polling a Periodic Stream with Burst

$R_{4,\text{data}}$, is required to finally determine the output stream $\mathcal{S}_{4,\text{out}}$ of the polling tasks.

### 6.6.1 Input Buffer

A task that periodically polls its input, implicitly performs shaping on the input stream. When both sender and receiver have the same period, polling corresponds to periodic shaping introduced in Section 6.2. Depending on the possible jitter or burst, it requires buffering and induces a delay, and the backlog and delay calculations from Equations 6.32 and 6.31 can be re-used. Such periodic task activation is often found in systems with *loosely coupled task communication*. The buffer synchronizes tasks which run independent schedules but still share a common periodicity or frequency as part of their combined function, e. g. in distributed filter functions.

When the timing of the receiver (its period) is nearly independent of the input data, the situation resembles key characteristics of sporadic shaping. Similar to condition 6.34, the polling task must be at least as fast as the input data arrives on stream $\mathcal{S}_{4,\text{data}}$ to prevent buffer overflows:

$$T_{4,\text{in}} \geq T_{4,\text{act}} \tag{6.56}$$

This requirement is based on the assumption that each input must essentially be read by the polling task. However, there are applications where this is not necessarily required. Simulink [115], for instance, defines periodic tasks that communicate through registers, and data can eventually be overwritten before it is read. This eliminates causal relations between event arrival and task execution. We have thoroughly analyzed the importance of register communication

and non-causal task execution in our SPI project (see Section 2.4). This thesis, however, focuses on causal performance dependencies, and register communication is not considered further.

The event curves of Figure 6.27 illustrate a polling scenario with bursts on the data input stream $\mathcal{S}_{4,\text{data}}$. The behavior in the case of polling is very similar to the behavior known from sporadic burst reduction, as introduced in Section 6.4.1. Hence, similar calculations can be applied. There is, however, a key difference. While shapers consume an input event as early as possible, a polling task must be assumed to read the input as late as possible, because it is executed independently of the input stream timing. In other words, an input event unusually has to *wait* until the next activation of the polling receiver. This adds a constant amount of time to the overall delay, and also affects buffering.

We can determine the end of the input burst and the corresponding two critical events using the known Equations 6.41 and 6.44 with some modifications. In Equations 6.47 and 6.48, we

- substitute the time-out value $d_{\mathcal{EAF},\text{out}}$ with the polling period $T_{4,\text{act}}$,
- use the more general $\delta_1^-(2)$ instead of $d_{\mathcal{EAF},\text{in}}$ to allow the application to jittery streams without burst, and
- add one activation period $T_{4,\text{act}}$ to the delay to account for the possible "late" consumption of the receiver

We obtain:

$$\begin{aligned} \mathit{backlog}^+_{4,\text{in}} = \max\Bigg( & 1 + \left\lfloor \frac{J_1}{T_1 - \delta_1^-(2)} \right\rfloor - \left\lceil \left\lfloor \frac{J_1}{T_1 - \delta_1^-(2)} \right\rfloor \frac{\delta_1^-(2)}{T_{4,\text{act}}} \right\rceil, \\ & 1 + \left\lceil \frac{J_1}{T_1 - \delta_1^-(2)} \right\rceil - \left\lceil \frac{\left\lceil \frac{J_1}{T_1 - \delta_1^-(2)} \right\rceil T_1 - J_1}{T_{4,\text{act}}} \right\rceil \Bigg) \end{aligned} \tag{6.57}$$

$$\begin{aligned} \mathit{delay}^+_{4,\text{in}} = T_{4,\text{act}} + \max\Bigg( & \left\lfloor \frac{J_1}{T_1 - \delta_1^-(2)} \right\rfloor (T_{4,\text{act}} - \delta_1^-(2)), \\ & \left\lceil \frac{J_1}{T_1 - \delta_1^-(2)} \right\rceil (T_{4,\text{act}} - T_1) + J_1 \Bigg) \end{aligned} \tag{6.58}$$

### 6.6.2 Execution Load and Response Time

The activation of the polling task is fully independent of the actual data input. Hence, scheduling analysis only needs to consider the activating event

stream $\mathcal{S}_{4,\text{act}}$. The execution time, which may be data-dependent, can be captured by an execution time interval. Scheduling analysis will then provide a corresponding response time interval and the minimum and maximum load. The details of the actual scheduling analysis are not in the key focus of this thesis. Rather, this work focuses on the amount of information required to enable the analysis. We see that the data input stream $\mathcal{S}_{4,\text{data}}$ is not essentially needed.

There are, however, scheduling analysis techniques that can distinguish between several execution behaviors or contexts [55]. For such techniques, the input stream $\mathcal{S}_{4,\text{data}}$ provides important information about the distribution of the two distinctive behaviors.

### 6.6.3 Output Event Streams

To determine the output stream, we have to know about the implementation of the polling task. There are two possible (and equally popular) ways to implement the output communication. One possibility is to always produce an output event and to code the success of the poll, or other state information, in the produced data. Such implementations are found in periodic data-flow implementations as generated by Simulink [115]. In this case, the situation can be reduced to a periodically activated task that always produces an output event, and the output calculations from Section 4.1 apply:

$$\begin{aligned} \mathcal{S}_{4,\text{out}} = \quad \mathcal{S}_{\mathsf{P+B}}\Big( \quad & T_{4,\text{out}} = T_{4,\text{act}}, \\ & J_{4,\text{out}} = J_{4,R} = R_4^+ - R_4^-, \\ & d_{4,\text{out}} = \max\left(T_{4,\text{act}} - J_{4,R}, R_4^-\right) \Big) \end{aligned} \tag{6.59}$$

The second possibility is to only produce an output event after a successful input poll, i. e. if a data input was available from stream $\mathcal{S}_{4,\text{data}}$. In other words, the task has a conditional output production. According to Section 4.5, such tasks produce their output sporadically. So we can directly re-use all calculations from Equation 6.59 but simply have to change the event model class from periodic into sporadic.

However, this procedure yields an overly conservative bound on the real output stream behavior. The polling task does not behave randomly but reacts to the availability of data on stream $\mathcal{S}_{4,\text{data}}$, and the actual input-output behavior is in fact deterministic. In other words, key properties of the input stream will be propagated to the output, such as an average period:

$$T_{4,\text{out}} = T_{4,\text{data}} = T_1 \tag{6.60}$$

We can account for this input-output relationship when we consider the entire path from task $\mathcal{T}_1$ to the output of task $\mathcal{T}_4$. Figure 6.28 illustrates this. We

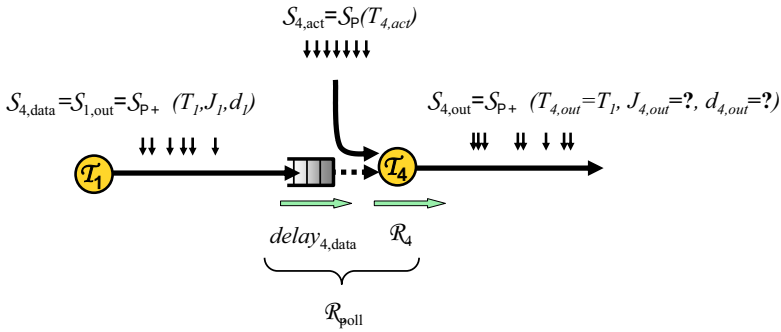



*Figure 6.28.* Output Stream of Conditional-Output Polling Tasks

determine the combined response time of the input buffer and the task execution:

$$\begin{aligned} R_{\text{poll}} &= delay_{4,\text{in}} + R_4 \\ R^+_{\text{poll}} &= delay^+_{4,\text{in}} + R^+_4 \\ R^-_{\text{poll}} &= delay^-_{4,\text{in}} + R^-_4 \\ J_{\text{poll}} &= R^+_{\text{poll}} - R^-_{\text{poll}} \end{aligned}$$

Now, we apply this combined response time to the input stream of task $\mathcal{T}_4$ and we obtain as output:

$$\begin{aligned} \mathcal{S}_{4,\text{out}} = \quad \mathcal{S}_{\text{P+B}}\Big( \quad & T_{4,\text{out}} = T_1, \\ & J_{4,\text{out}} = J_1 + J_{\text{poll},R}, \\ & d_{4,\text{out}} = \max\left(T_1 - J_{\text{poll}}, R^-_4\right) \Big) \end{aligned} \tag{6.61}$$

We can even incorporate key information about the activation period of task $\mathcal{T}_4$ to further tighten the minimum output distance $d_{4,\text{out}}$ by using the maximum value of both Equations 6.59 and 6.61:

$$d_{4,\text{out}} = \max\left(T_1 - J_{\text{poll}}, T_{4,\text{act}} - J_{4,R}, R^-_4\right) \tag{6.62}$$

This is specifically useful, if the input stream is bursty and the buffering delay exceeds the original period. This optimization accounts for the fact that

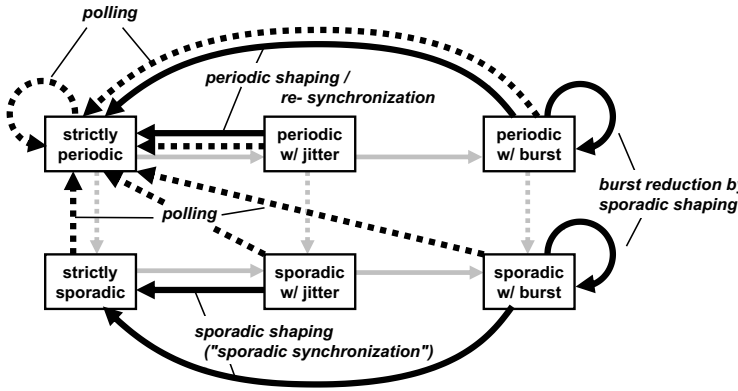



*Figure 6.29.* Existing Event Adaptation Functions

periodic polling shares key properties with sporadic shaping, as introduced in Section 6.4. It buffers input and reduces transient frequencies.

Clearly, periodic polling can be applied to sporadic streams as well. The output stream of $\mathcal{T}_4$ is a sporadic stream with the properties provided by Equations 6.61 and 6.62.

## 6.7 Summary

We have seen that the timing of events in a stream can be controlled by means of buffers and timers in order to modify specific properties of a stream such that new model transformations are enabled, in addition to those of the previous chapter. Such event adaptation functions improve the usability and compatibility of standard event models in flow-based scheduling analysis further.

Two fundamentally different shaping techniques have been introduced. The event model transformations that have been enabled by these two shaper types are illustrated as bold arrows in Figure 6.29. Periodic shaping is used to re-synchronize periodic streams with jitter and burst to their original period. Sporadic shaping reduces transient burst frequencies for all stream types, and allows to virtually re-synchronize sporadic streams. The buffer requirements and delays that shaping introduces have been thoroughly investigated and formal frameworks for calculating bounds on delays and backlogs have been developed.

We have carried out an experiment, which illustrates that the impact of shaping depends heavily on the choice of the shaper. By varying the shaping type,

periodic or sporadic, and the time-out value, designers can deliberately control and optimize the system behavior. We call such optimizations *user-controlled shaping* that we distinguish from *automatic shaping*, which allows the systematic reduction of shaping backlogs and delays, whilst still meeting given input requirements.

Finally, we have analyzed the situation of input polling. Polling is basically an implementation choice for a task rather than an operation on an event stream. The influence of polling on buffers and event timing, however, is strongly related to shaping. Depending on the specific situation, polling can be formally seen as either type of shaping, and the derived formal framework allows calculation of buffer sizes and delays also for systems with polling. We have shown that, and how, the event stream view helps to optimize the output stream accuracy in the presence of polling.

Chapter 7

# SYSTEM-LEVEL ANALYSIS PROCEDURE

Each of the four preceding chapters provides one important concept that lets us a) capture component inputs and task activation as a pre-requisite for scheduling analysis, b) model and provide a relation between inputs and outputs of a task or a component, influenced by scheduling and execution, c) provide appropriate event model interfaces to match the analysis requirements, and d) apply appropriate event stream adaptation to facilitate the interfacing step (or to optimize the scheduling).

These concepts cover the key local aspects of scheduling and response time analysis in a system-level context. This chapter integrates these individual concepts and uses them as *hand tools* to apply the flow-based analysis approach, as introduced in Section 2.3, to systems with multiple heterogeneous event models.

We start with a brief review of the introduced formalisms and define an appropriate system analysis model. In order to extend the iterative flow-based approaches from Thiele [116] and Gresser [38, 39] to support heterogeneous event models, we have to insert an appropriate event model interfacing step. We will see that these interfaces not only enable this new analysis procedure, but that adaptations also allow control of the streams during the analysis and integration process, and to globally optimize the system. This is of particular importance in systems where the mixture of architectural and functional influences leads to complex and confusing cyclic dependencies and scheduling anomalies. These can turn the iterative analysis into a convergence problem, and we identify monotonicity properties to formulate termination conditions. We show how bottlenecks and pitfalls can be identified, resolved, and optimized using this new analysis procedure. We use expressive examples to demonstrate the individual steps as well as the overall analysis.

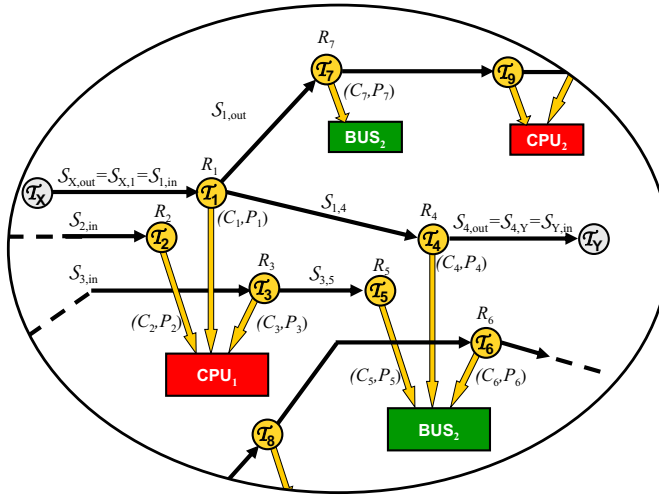



*Figure 7.1.* System Model Example

## 7.1 System Analysis Model

We summarize the required elements, parameters, and the overall structure using part of an example system as illustrated in Figure 7.1.

### 7.1.1 Model Elements and Parameters

The system consist of a set of tasks $\tau_i$. Each task $\tau_i$ is mapped to a resource $\mathsf{RSC_j}$ and is scheduled there according to the resource's scheduling strategy. Each task has two mapping-specific parameters: the core task time $C_i$ and the scheduling parameters $P_i$. We do not further distinguish between computation tasks (processes) and communication tasks (channels, links, etc.). Both require a net resource access time ($C_i$). The actual response time $R_i$ defines the time between the occurrence of the request (activation time) and the time this request has been successfully served (completion). Both the core task time and the response time are intervals with upper and lower bounds.

### 7.1.2 Model Structure and Dependencies

The response time $R_i$ abstracts from the detailed scheduling influences that task $\tau_i$ experiences due to e. g. preemptions by other tasks. It represents the tasks externally observable execution or communication timing behavior. The interaction with other tasks is captured using event streams: an input event

stream $\mathcal{S}_{i,\text{in}}$ and an output event stream $\mathcal{S}_{i,\text{out}}$. Through these uni-directional streams, tasks communicate with each other. The output stream $\mathcal{S}_{i,\text{out}}$ of task $\mathcal{T}_i$ becomes the input stream $\mathcal{S}_{k,\text{in}}$ of task $\mathcal{T}_k$, the stream itself is denoted $\mathcal{S}_{i,k}$. Each task must have exactly one input stream, while a task's output can be connected to several task's inputs. In the example of Figure 7.1, the output of $\mathcal{T}_1$ is connected to two tasks, $\mathcal{T}_7$ and $\mathcal{T}_4$, respectively. This allows task trees to be captured.

### 7.1.3 Environment Modeling

Tasks $\mathcal{T}_X$ and $\mathcal{T}_Y$ are *environmental tasks* that model timing constraints and assertions of the environment that the system is embedded in. Task $\mathcal{T}_X$ is an environmental *source* task that provides information about the timing of the primary inputs to the system, such as an actuator connected to $\mathsf{CPU_1}$. $\mathcal{T}_Y$ is an environmental *sink* task that also has a certain timing behavior representing a *timing constraint*, such as a maximum bound on the allowed jitter or a maximum sporadic period. Using event streams and models, not only *within* the system but also between the system and its environment, we can elegantly model environmental assertions and constraints without requiring additional concepts.

### 7.1.4 Local Scheduling

The event streams are the key integrating elements in this proposed analysis approach. Each resource $\mathsf{RSC_j}$ has one local analysis method associated that calculates the response time $R_i$ of the tasks as reviewed in Section 2.1. The analysis takes as input the following parameters of each task $\mathcal{T}_i$ mapped on that resource: the core task time $C_i$, the scheduling parameter $P_i$, and the *activating event stream* $\mathcal{S}_{i,\text{act}}$.

The analysis requires the activating stream to comply with an analysis-dependent *input event model requirement* $\mathcal{R}_{j,\text{in}}$ as introduced in Chapter 5. In order to meet this requirement, the actual input stream $\mathcal{S}_{i,\text{in}}$ might need to be appropriately transformed by means of event model interface and/or automatic shapers as introduced in Chapters 5 and 6.

After analyzing the scheduling, the output event stream $\mathcal{S}_{i,\text{out}}$ is determined from the input stream and the response time using a function $\mathcal{O}(R_i, \mathcal{S}_{i,\text{in}})$. We thoroughly discussed the dependencies between response times and input and output streams in Chapter 4.

## 7.2 The System-Level Analysis Procedure

Figure 7.2 provides an abstract view of the overall system-level analysis procedure consisting of 7 steps and a set-up step (step 0). The overall idea of iterative event stream propagation is not fully new, it has been proposed

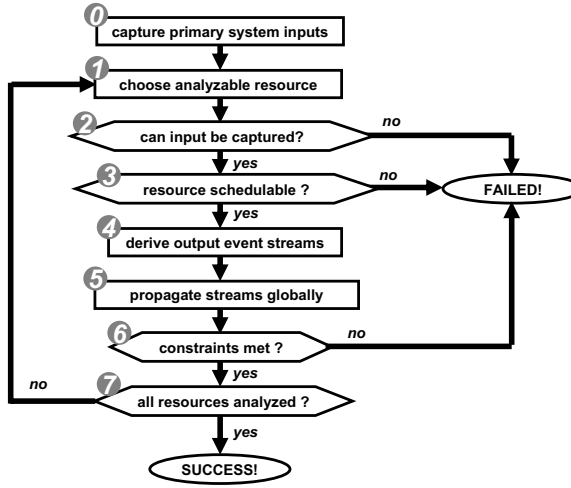



*Figure 7.2.* System-Level Analysis Procedure

by Gresser [38, 39] and Thiele et. al. [116]. The novelty of our approach is that is allows the use of known local analysis techniques, that possibly require heterogeneous event models, enabled by event model interfacing and stream adaptation.

### Step 0: Capturing Primary System Inputs

The primary system inputs are represented as environmental source tasks. We introduced environmental modeling in Section 7.1.3. The environment specification determines the input event streams of the directly connected tasks:

$$\forall \mathcal{S}_{a,b}, a \in \{\mathcal{T}_{\text{env}}\} : \mathcal{S}_{b,\text{in}} = \mathcal{S}_{a,\text{out}} \tag{7.1}$$

### Step 1: Choosing an Analyzable Resource

Scheduling analysis requires the activation of all tasks to be known. Hence, we can only analyze a resource when the activating event streams are known, in turn requiring that the input streams are known for all tasks on that resource. In the beginning, these are usually the ones which are connected to the primary system inputs.

### Step 2: Capturing Task Activation

After selecting a resource for analysis, we have to capture the activation of all tasks in a way suitable for the analysis. This might impose an analysis-specific

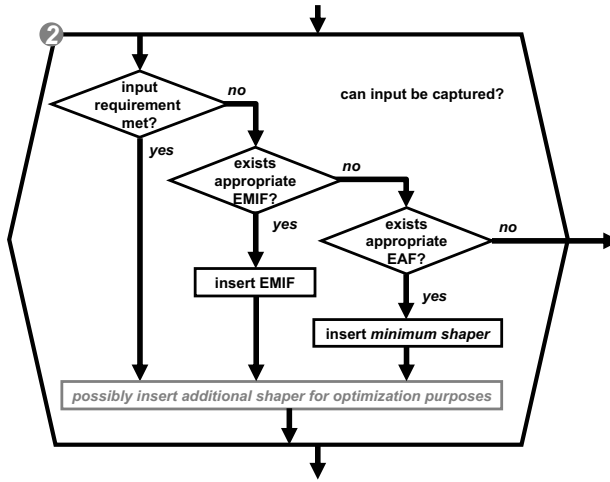



*Figure 7.3.* Input Event Stream Capturing

*input model requirement* $\mathcal{R}_{i,\text{in}}$ on the task input streams. Depending on the situation, this step might incorporate several sub-steps illustrated in Figure 7.3. We have to apply the procedure to each task separately.

We start with testing whether the input stream meets the requirement (see Section 5.2):

$$\mathcal{S}_{i,\text{in}} \stackrel{?}{\in} \mathcal{R}_{i,\text{in}} \tag{7.2}$$

If the test is successful, the input stream already represents the activating stream that can be used for scheduling analysis (in step 3):

$$\mathcal{S}_{i,\text{act}} = \mathcal{S}_{i,\text{in}} \tag{7.3}$$

If test 7.2 fails, we look for an *event model interface* that transforms the input stream representation into a representation suitable for analysis. We apply the requirement coverage test from Definition 5.4:

$$\mathcal{RC}(\mathcal{S}_{i,\text{in}}, \mathcal{R}_{i,\text{in}}) \stackrel{!}{=} 1 \text{ (true)} \tag{7.4}$$

If the test is successful, we can insert the appropriate $\mathcal{EMIF}$s introduced in Chapter 5 to transform the input stream into an activating stream that meets the input requirement:

$$\mathcal{S}_{i,\text{act}} = \mathcal{EMIF}_{\mathcal{EM}_{i,\text{in}} \rightarrow \mathcal{EM}_{i,\text{req}}}(\mathcal{S}_{i,\text{in}}) \in \mathcal{R}_{i,\text{in}} \tag{7.5}$$

If also test 7.4 fails, we try inserting an *automatic shaper* (see Definition 6.1), that reduces too high transient loads or fully eliminates jitters. Then, we check if the resulting shaped stream meets the input requirement, possibly with an additional $\mathcal{EMIF}$:

$$\mathcal{S}_{i,\text{shaped}} = \mathcal{EAF}_{\text{auto}}(\mathcal{S}_{i,\text{in}}, \mathcal{R}_{i,\text{in}})$$
$$\mathcal{RC}(\mathcal{S}_{i,\text{shaped}}, \mathcal{R}_{i,\text{in}}) \stackrel{!}{=} 1(\text{true})$$

When even automatic shaping does not help, the input stream can simply not be reasonably transformed into the required representation. In effect, (at least some of) the tasks on that resource will not be analyzable by the next step, and we can terminate the iterative analysis procedure. There are two classes of reasons for such failures: First, to prevent inputs that lead to permanent overload, a task might have a minimum allowed average period, reflecting a maximum average execution frequency. Input streams that have a smaller average period are not acceptable. Second, there might be a general mismatch between the intended behavior of the producer task and the consumer task, for instance both have different periods but were assumed to execute in strong relationship with each other. Either source of failure represents a well-known system integration problem. The event stream view implicitly detects such integration problems.

In case of a successful activation modeling, i. e. all input streams can be transformed to meet the corresponding task input requirements, a user might consider additional local optimizations using periodic or sporadic shaping. We have shown the impact of such *user-controlled shaping* in Section 6.4.3. A special case where such local optimizations help to globally increase the schedulability of the system, is discussed in Section 7.7.

### Step 3: Local Scheduling Analysis

When all task activation models are known from the previous step, the scheduling on the selected resource can be analyzed. For local scheduling analysis, a big variety of techniques is available from the real-time systems community. We reviewed popular techniques and landmark publications in Section 2.1. In addition to the task activation models, these techniques require the core execution time $C_i$ as well as the scheduling parameters $P_i$ of each task to be known. The scheduling analysis returns the response time $R_i$ of each task which determines the externally observable timing behavior of a task.

This is the second step of the overall procedure that can fail. An example is a permanent overload situation that clearly makes some, if not all tasks unschedulable. Less obvious examples are situations where the system is in fact schedulable but the analysis has limitations that prevent it from calculating a bounded response time. The initial proposal for rate-monotonic analysis [73]

is such an example. Response times larger than periods are just not considered. In either case, the local analysis cannot determine a response time for a subset of tasks. Again, the systematic procedure and the event stream view help in identifying a potential modeling insufficiency or a performance bottleneck.

### Step 4: Determining the Output Streams

We have shown in Chapter 4 that the output stream is a function of the input stream and the response time:

$$\mathcal{S}_{i,\text{out}} = \mathcal{O}(\mathcal{S}_{i,\text{in}}, R_i)$$

If available, additional information about the implementation of a task, for instance polling characteristics (see Section 6.6), can further tighten the output stream calculations.

### Step 5: Propagating Event Streams Globally

In this step, the just obtained output streams are propagated to the inputs of the connected tasks in the application:

$$\mathcal{S}_{k,\text{in}} = \mathcal{S}_{i,\text{out}}$$

### Step 6: Constraint Verification

When all tasks have been successfully analyzed, the system is schedulable, in general meaning that no permanent overload conditions occurred, and we can check if the given constraints are met. We have already mentioned that environmental sink tasks can be used to model constraints on event steams that enter the environment, and there is a variety of additional possibilities to verify other system properties such as local deadlines, buffer constraints and path latencies, which we explain in Section 7.3. If one of these constraints is violated, the analysis can be aborted.

### Step 7: Iteration

Now, the whole process iterates from the beginning. During the iteration, event streams are propagated through the system of locally analyzable components, from the primary system inputs to the system outputs, i. e. to the environmental sink tasks. The iteration is successfully completed when all resources have been analyzed, and fails if either not all task inputs could be appropriately transformed to meet requirements, or a bounded response time could not be determined for all tasks, or another constraint is violated.

## 7.3 Local and Global Constraint Verification

### 7.3.1 Environmental Constraints

A dedicated period and a maximum allowed jitter at the interface between the system outputs and the environment is a popular example for an environmental constraint. It can be elegantly modeled using a virtual sink task with an appropriate input requirement:

$$\mathcal{R}_{\text{sink,in}} = \{\mathcal{S}_{\mathsf{P+J}} \in \mathcal{EM}_{\mathsf{P+J}} | T = T_{\text{sink}}, J \leq J_{\text{sink}}\}$$

As another example, a maximum allowed frequency can be modeled using a sporadic input requirement. As they use virtually the same modeling concepts, the environmental source and sink tasks can be included in the analysis procedure, and the requirements are automatically checked during step 2. Moreover, stream transformation using $\mathcal{EMIF}$s and $\mathcal{EAF}$s is considered representing an *automatic output adaptation*. However, in our examples later, this particular check is delayed and considered after the system has been fully analyzed.

### 7.3.2 Internal Event Streams

The event streams capture in detail the interaction between tasks and provide information about the resource access patterns of individual tasks. The additional delays and the buffer memory that is required because of shaping allows the impact of such shaping decisions on the overall system timing and performance to be determined.

### 7.3.3 Local Task and Resource Properties

The local response times as a direct result from the individual task scheduling analysis allow the actual execution timing to be checked against the expected timing and/or local constraints such as a deadline. It furthermore provides information about each individual task's performance for a given mapping of tasks to resources.

The total load or utilization of each resource in the system allows the identification of bottlenecks, as well as the remaining performance in the architecture. In practice, this can be of particular importance for selecting components (resources) to be optimized in terms of speed and performance, e. g. bus bandwidth.

### 7.3.4 Path Constraints

From the mentioned local timing properties we can determine key global timing and performance properties. Very often, it is not the individual tasks that are constrained to exhibit a certain timing behavior. Rather, the interactions of several tasks form a system function that is required to meet a certain deadline.

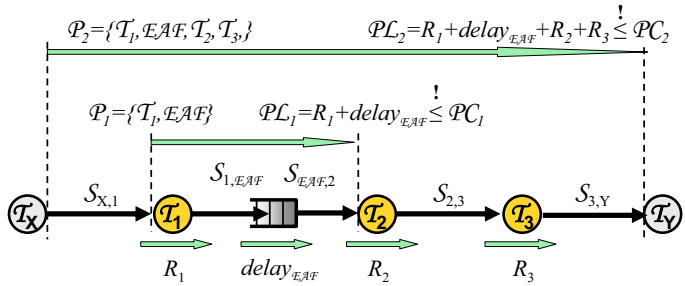



*Figure 7.4.* Path Latencies and Constraints

Such system functions usually span a whole *path* from primary system inputs to primary system outputs, including several tasks and event streams.

**Definition 7.1.** ***(Path)***
*A path $\mathcal{P}_i$ is a connected set of tasks, shapers, and event streams. A path must have a dedicated starting element, and a termination element, and at most one output event stream of each task can be included in that path, prohibiting a path from containing branches.* ■

**Definition 7.2.** ***(Path Constraint)***
*A Path Constraint $\mathcal{PC}_i$ is a positive real that bounds the allowed maximum time each event requires to travel along that path $\mathcal{P}_i$.* ■

To check the compliance with such path constraints, we need to determine the actual path latency or path delay by accumulating the delays of the individual path elements.

**Definition 7.3.** ***(Path Latency)***
*The accumulated task response times and shaping delays of all elements in a path $\mathcal{P}_i$ yield the Path Latency $\mathcal{PL}_i$:*

$$\mathcal{PL}_i = \sum_{\mathcal{T}_j \in \mathcal{P}_i} R_{\mathcal{T}_j} + \sum_{\mathcal{EAF}_k \in \mathcal{P}_i} delay_{\mathcal{EAF}_k} \tag{7.6}$$

*The path latency is an interval, and the upper and lower bounds are determined by accumulating the corresponding upper and lower bounds of the response times and shaping delays.* ■

Figure 7.4 shows three system tasks, one sink and one source task, and one shaper, which are connected as a path with five event streams. Two paths $\mathcal{P}_1$ and $\mathcal{P}_2$ are shown together with their respective latencies. These paths

represent chains in the application and could possibly be specified by their start and end node –task or shaper. Other work allows also functional cycles to be included in a path [140, 52], that are not considered further here.

### 7.3.5 Buffer Constraints

The execution backlog of all tasks on one resource determines (together with the "size" of the data that the events might carry) the total required input buffers which might be constrained because of limited memory for that resource. Likewise, the shaping backlog of all elements provides the memory required for shaping. We mentioned that such shaping can be elegantly done by operating system functions or hardware drivers that adds to the overall required memory. The required execution and shaping backlogs along a path also provide interesting information about the "bottleneck distribution" along a path.

## 7.4 Cyclic Event Stream Dependencies

So far, we have assumed that we can always find a resource with all input streams known in each analysis iteration (step 1). Unfortunately, this is not necessarily the case. Recall the example system from the introduction. We highlighted a non-functional dependency cycle in Figure 1.5. The two key components, the CPU and the bus, are mutually dependent on each other with respect to the event streams. Each resource requires an output stream of the other one to be known before it can be analyzed. In such situations the primary system inputs do not provide a starting point for the analysis procedure of Section 7.2, since no resource has all input streams known.

It is important to recognize that such dependency cycles can turn the iterative analysis procedure into a *convergence problem* as illustrated in Figure 7.5, and we are facing two key challenges: We have to find a reasonable *starting point* for the iterative procedure and we have to guarantee *convergence*, or find a reasonable *termination condition*. The detailed analysis of the impact of scheduling and execution on output event streams in Chapter 4 provides us with a basic approach.

### 7.4.1 The Starting Point

We know that execution and scheduling *adds* jitter characteristics to event streams from task inputs to task outputs, i. e. the jitter grows from input to output at each local analysis while the original input period is preserved at the output, regardless whether periodic or sporadic. We can exploit this behavior and start the actual analysis procedure by propagating all input streams from the primary system inputs through the system *without* considering the influence of scheduling. In other words, we simply "copy" the input stream of a task to

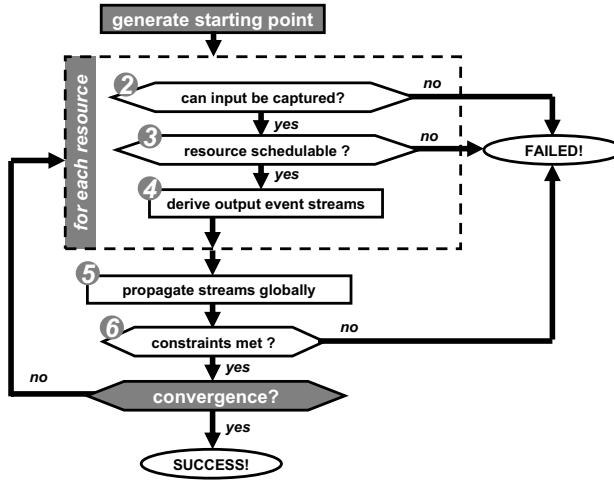



*Figure 7.5.* System-Level Analysis Turns into a Convergence Problem

its output, and we do this for all tasks. This is the stating point for the cyclic analysis.

### 7.4.2 Cyclic Analysis

In the first *analysis cycle*, we analyze each resource exactly once. More precisely, we apply the steps 2, 3, and 4 to all resources, on after another (see Figure 7.5). In that analysis cycle, the response time interval of all tasks are determined. If the response time interval of each task has identical upper and lower bounds, i. e. the response time would be constant, then the output jitter would equal the input jitter, which was already given by the starting point. In other words, the analysis would have already converged.

If at least one task has a non-constant response time, then the width of the response time interval determines the new output jitter of that task. This invalidates the initial starting-point assumptions. Consequently, we propagate the output streams of all tasks to the inputs of all connected tasks (step 5), check if the constraints are met (step 6), and have to perform a second analysis cycle, that re-analyzes the tasks on all resources whose input jitters have changed.

After the second analysis cycle, we have to check again, whether the output jitters have increased, and possibly perform another analysis cycle, and so on. The analysis is completed when the cyclic procedure converges, i. e. no output jitter values increase anymore. The entire cyclic procedure including the con-

vergence test is illustrated in Figure 7.5. The important question is: *Does the analysis converge, and when?*

### 7.4.3 Convergence and Termination

When we take a closer look on the consequences of increasing input jitter of a task, we can distinguish two effects. First, the output jitter of that task will also increase because of the *jitter inheritance* phenomenon, that we mentioned in Section 4.1.3. This inheritance applies to all tasks along paths in the system. The second effect of increasing input jitters is *scheduling distortion*. An increasing input jitter distorts the scheduling and potentially increases the amount of worst-case interference, consequently increasing worst-case response times. Conversely, the best-case response times decrease, finally resulting in larger output jitters. This also shows that our initial assumption is valid, since the initial input streams with smaller jitters are *covered* by all later input streams with a possibly larger jitter. In other words, the initially assumed system behavior is still included in the possible behavior of all later steps.

Convergence criteria and the actual time-complexity of the entire analysis, however, are extremely complex to determine, even though we know the influences that let jitters and response times increase. We could at best bound the number of analysis cycles, but the overall complexity also depends strongly on the complexity of the local analysis algorithms. These are embedded in the global cyclic analysis procedure and could possibly pose further questions to numerical convergence and stability. The holistic analysis approach (see Section 2.2.2), that seeks an iterative solution of a complex set of recursive system-level timing equations, is faced with a comparable convergence problem. This, however, has not even been solved there, although many groups continuously extend and enhance the related holistic equations and algorithms. In this thesis, we will not investigate this issue further, since we can formulate a reasonable termination condition that allows the efficient application of the cyclic analysis procedure in practice.

We know that, in case of fixed-priority scheduling, the interference of a task is the sum of integer multiples of the core execution time of other tasks on the same resource (see Equation 3.22). Likewise, in time-driven scheduling, the interference is an integer multiple of one or more slot times (see Equation 2.9), etc.. Consequently, the amount of interference can only increase in discrete steps; and the minimum increment is bounded by, for instance, the minimum core execution time, the smallest slot time, or the execution time of some operating system function. In effect, the response times and the output jitters of at least some tasks will monotonically grow with a minimum “speed”. This minimum speed prevents the jitters and response times from asymptotic expansion. Sooner or later, the analysis will either successfully converge, or a constraint will be violated, and the analysis can be safely terminated. This lets us bound

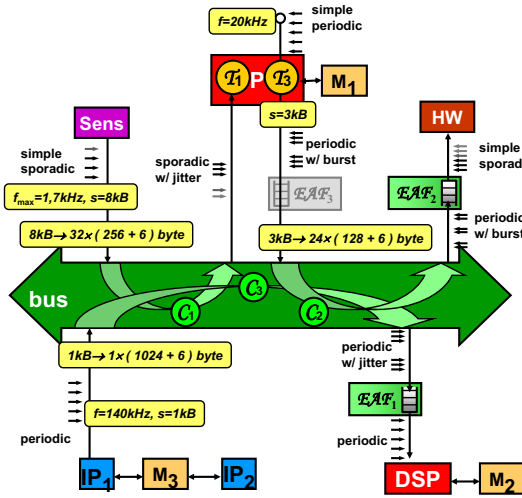



*Figure 7.6.* Detailed System Example

the number of analysis cycles from the given deadlines, path constraints, environmental sink constraints, and task input requirements. As an additional advantage, the global solution procedure proposed tightly follows the event- and data-flow in the system. This makes the cyclic analysis, especially with respect to convergence issues, easily comprehensible.

## 7.5 Example

Now, we demonstrate the applicability of the approach by fully analyzing the example from the introduction (see Figure 1.2) where we have integrated two functionally independent sub-systems using a shared bus. The CPU sub-system consists of a sensor that eventually sends data that must be processed by the CPU. In turn, the CPU accesses a hardware accelerator (HW). The DSP sub-system runs a highly optimized digital signal processing application and the communication over the bus is critical. We have highlighted the key analysis challenges in Section 1.1 including cyclic dependencies and model incompatibilities.

### 7.5.1 System Set-Up

Figure 7.6 provides a detailed view on the entire system. The individual components are now specified.

#### 7.5.1.1 The Primary System Inputs

There are three inputs to the system: The sensor sporadically sends data blocks of $8kb$ size to task $\mathcal{T}_1$, with a maximum sending frequency of $1,7kHz$, which corresponds to a *sporadic event model* with a minimum sporadic period of $\frac{1}{1,7kHz} = 588\mu s$. Task $\mathcal{T}_3$ is periodically activated by an RTOS (real-time operating system) timer on the CPU with a period of $\frac{1}{20kHz} = 50\mu s$. The high-performance DSP application on the $\mathsf{IP}_1$ component has a sending frequency of $140kHz$, corresponding to a period of $7,14\mu s$.

These inputs are modeled as environmental source tasks. The formal event stream definitions are:

$$\begin{aligned} \mathcal{S}_{\mathsf{sens}} &= \mathcal{S}_{\mathsf{S}}(T_{\mathsf{sens}} = 588.2) \\ \mathcal{S}_{\mathsf{timer}} &= \mathcal{S}_{\mathsf{P}}(T_{\mathsf{timer}} = 50) \\ \mathcal{S}_{\mathsf{IP}} &= \mathcal{S}_{\mathsf{P}}(T_{\mathsf{IP}} = 7.14) \end{aligned}$$

#### 7.5.1.2 The Bus Timing

The communication over the bus is implemented using a non-preemptive static priority protocol. Instead of sending the complete data blocks as a single transmission, the data is fragmented into smaller packets to avoid extended *blocking times* (see Section 2.1.2.4). Each $8kB$ data block from the sensor is split into 32 packets of $262byte$ each, $256bytes$ plus $6bytes$ protocol overhead–address, length, and CRC. The $3kB$ blocks from task $\mathcal{T}_3$ are split into 24 packets of $(128{+}6) = 134bytes$. This channel $\mathcal{C}_2$ has a higher priority than channel $\mathcal{C}_1$. The highest-priority channel $\mathcal{C}_3$ does not split the DSP data packets, but only adds the $6byte$ protocol information.

The actual network speed is $300Mbyte/s$. Hence, the actual core communication times are:

$$\begin{aligned} C_{\mathcal{C}_1} &= 32 \times (256+6)byte \times \frac{1s}{300Mbyte} = 27.95\mu s \\ C_{\mathcal{C}_2} &= 24 \times (128+6)byte \times \frac{1s}{300Mbyte} = 10.72\mu s \\ C_{\mathcal{C}_3} &= 1 \times (1024+6)byte \times \frac{1s}{300Mbyte} = 3.43\mu s \end{aligned}$$

Because the communication of an individual packet cannot be preempted, the higher-priority channels can experience blocking by lower-priority communications. The longest packets that can block other communications are sent by the sensor, hence it determines the blocking times of the other channels:

$$B_{\mathcal{C}_2} = B_{\mathcal{C}_2} = (256 + 6)byte \times \frac{1s}{300Mbyte} = 0.87\mu s$$

The overall average network usage $U_{\text{net}}$ is:

$$\begin{aligned} U_{\text{net}} &= U_{\mathsf{sens}} + U_{\mathsf{CPU}} + U_{\mathsf{DSP}} \\ &= 1,7kHz \times 32 \times 262byte + 20kHz \times 24 \times 134byte \\ &\qquad +140kHz \times 1 \times 1030byte \\ &= 14,2528Mbyte/s + 64,32Mbyte/s + 144,2Mbyte/s \\ &= 222,77Mbyte/s \end{aligned}$$

With respect to the actual bus speed, the utilization is:

$$\mathcal{U}_{\text{net}} = \frac{U_{\text{net}}}{300Mbyte/s} = 74,3\% \tag{7.7}$$

#### 7.5.1.3 The Task Core Execution Timing

For simplicity, the core execution times of the two tasks on the CPU are assumed constant. This is not a restriction of the local analysis, nor of the overall global analysis approach, but it simplifies the discussion of the results:

$$\begin{aligned} C_{\mathcal{T}_1} &= 250 \\ C_{\mathcal{T}_3} &= 10 \end{aligned}$$

In addition, we model the operating system influence using blocking times for both tasks $\mathcal{T}_1$ and $\mathcal{T}_3$:

$$\begin{aligned} B_{\mathcal{T}_1} &= 15 \\ B_{\mathcal{T}_3} &= 15 \end{aligned}$$

#### 7.5.1.4 Constraints

The hardware block HW as well as the signal processor (DSP) are imposing constraints on their input timing. Due to its limited execution speed, the frequency of the input to the hardware block is constrained by:

$$f_{\mathsf{HW,max}} = 50kHz = \frac{1}{20\mu s}$$

The DSP requires fully periodic inputs in order to run a set of signal processing applications with a highly optimized cyclic schedule. The frequency equals the

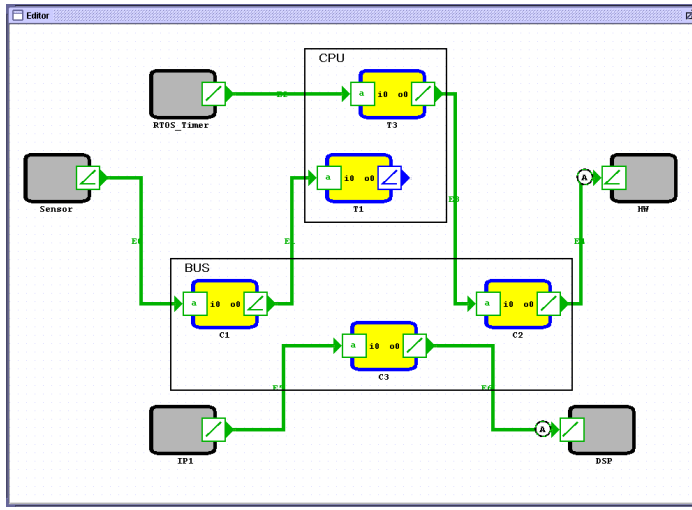



*Figure 7.7.* SymTA/S Model of Example System

sending frequency of the IP sub-system, since both components were initially directly connected (see Figure 1.2):

$$f_{\mathsf{DSP,in}} = 140kHz = \frac{1}{7.14\mu s}$$

These constraints translate into input model requirements (see Section 5.2) for the two corresponding tasks:

$$\begin{aligned} \mathcal{R}_{\mathsf{DSP,in}} &= \{\mathcal{S}_{\mathsf{P}}(T_{\mathsf{DSP}} = T_{\mathsf{IP}} = 7.14)\} \subset \mathcal{EM}_{\mathsf{P}} \\ \mathcal{R}_{\mathsf{HW,in}} &= \{\mathcal{S}_{\mathsf{S}}(T_{\mathsf{HW}}) | T_{\mathsf{HW}} \geq 20)\} \subset \mathcal{EM}_{\mathsf{S}} \end{aligned}$$

### 7.5.2 Analysis Set-Up

To perform the analysis, we use the SymTA/S tool which we have developed in our group. Figure 7.7 shows the example system in the editor window of the tool. Environmental tasks have a gray body, while scheduled tasks that must be analyzed are yellow (and turn white after successful analysis). The icons in the task ports abstractly illustrate the general shape of the $\eta(\Delta t)$ event curves and provide a quick visualization about the event model, in which the stream is represented.

The specific properties of the sensor and the IP sub-system are captured as environmental source tasks, and the HW block as well as the DSP sub-system as environmental sinks. An output port icon denotes the event model in which the actual output stream is provided: periodic for the IP component and the RTOS timer, sporadic for the sensor. An input port icon visualizes an event stream requirement: periodic for the DSP and sporadic for the HW component. The character "a" in the input ports of the tasks and channels show that the analysis can cope with *a*ny input event model, i. e. there is no specific input model requirement. The system has just been initialized, i. e. a starting point has been generated according to Section 7.4.1 by propagating the event streams from the task inputs to task outputs.

**Local Analysis Technique**

For the local scheduling analysis on both the CPU and the bus, we have slightly modified the analysis technique from Lehoczky [69] and Tindell [121] (see also Sections 2.1.2.2 and 3.3.2) to support our six-class event model set. For the best-case response time, we use an approximation algorithm [90].

## 7.6 Iterative Analysis

We will now apply our novel analysis procedure to the given system. We will perform several analysis cycles until all event streams –especially the jitters– converge, and we shall report in detail on the results of each individual cycle.

### 7.6.1 Analysis Cycle 0: Starting Point Generation

As introduced in Section 7.4.1, we first propagate all event streams from environmental sources through the system following the task dependencies in order to generate a reasonable starting point for the iterative analysis procedure:

$$
\begin{aligned}
\mathcal{S}_{\mathcal{T}_1,\text{in}}{}^{(0)} &= \mathcal{S}_{\mathcal{C}_1,\text{out}}{}^{(0)} = \mathcal{S}_{\mathcal{C}_1,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{sens}} = \mathcal{S}_{\mathsf{S}}(588.2) \\
\mathcal{S}_{\mathsf{DSP},\text{in}}{}^{(0)} &= \mathcal{S}_{\mathcal{C}_3,\text{out}}{}^{(0)} = \mathcal{S}_{\mathcal{C}_3,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{IP}} = \mathcal{S}_{\mathsf{P}}(7.14) \\
\mathcal{S}_{\mathsf{HW},\text{in}}{}^{(0)} &= \mathcal{S}_{\mathcal{C}_2,\text{out}}{}^{(0)} = \mathcal{S}_{\mathcal{C}_2,\text{in}}{}^{(0)} = \mathcal{S}_{\mathcal{T}_3,\text{out}}{}^{(0)} = \mathcal{S}_{\mathcal{T}_3,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{timer}} = \mathcal{S}_{\mathsf{P}}(50)
\end{aligned}
$$

### 7.6.2 Analysis Cycle 1

We apply these event streams of step 0 as the actual inputs of step 1 and analyze the CPU and the bus independent from each other.

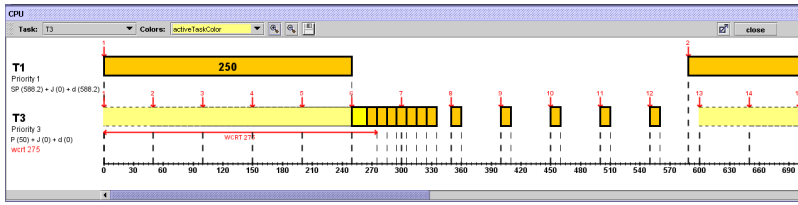



*Figure 7.8.* CPU Scheduling Diagram of First Analysis Cycle

#### 7.6.2.1 CPU Analysis

The input streams of the two tasks on the CPU are:

$$\begin{aligned} \mathcal{S}_{\mathcal{T}_1,\text{in}}{}^{(1)} &= \mathcal{S}_{\mathcal{T}_1,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{S}}(588.2) \\ \mathcal{S}_{\mathcal{T}_3,\text{in}}{}^{(1)} &= \mathcal{S}_{\mathcal{T}_3,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{P}}(50) \end{aligned}$$

The local analysis of Section 7.5.2 does not impose any input event model requirements, so we can directly use the input streams to capture task activation (step 2) and perform the local analysis (step 3). This yields the following response times:

$$R^{+}_{\mathcal{T}_1}{}^{(1)} = 265 \qquad R^{-}_{\mathcal{T}_1}{}^{(1)} = 250$$

$$R^{+}_{\mathcal{T}_3}{}^{(1)} = 275 \qquad R^{-}_{\mathcal{T}_3}{}^{(1)} = 10$$

The scheduling diagram in Figure 7.8 shows the critical instant of task $\mathcal{T}_3$. Due to the preemption by $\mathcal{T}_1$ and the additional blocking time $B_{\mathcal{T}_3}$ the sixth activating event arrives before the first execution starts. This leads to a bursty execution behavior of task $\mathcal{T}_3$ and an *execution backlog* that must be appropriately buffered. The execution backlog of task $\mathcal{T}_1$ is one:

$$\mathit{backlog}_{\mathcal{T}_1}{}^{(1)} = 1 \qquad \mathit{backlog}_{\mathcal{T}_3}{}^{(1)} = 6$$

With the functions defined in Chapter 4, we determine the task output event streams (step 4):

$$\begin{aligned} \mathcal{S}_{\mathcal{T}_1,\text{out}}{}^{(1)} &= \mathcal{S}_{\mathsf{S+J}}(T_{\mathcal{T}_1,\text{in}}{}^{(1)}, J_{\mathcal{T}_1,R}{}^{(1)}) \\ &= \mathcal{S}_{\mathsf{S+J}}(588.2, 15) \\ \mathcal{S}_{\mathcal{T}_3,\text{out}}{}^{(1)} &= \mathcal{S}_{\mathsf{P+B}}\left(T_{\mathcal{T}_3,\text{in}}{}^{(1)}, J_{\mathcal{T}_3,R}{}^{(1)}, \max(T_{\mathcal{T}_3,\text{in}}{}^{(1)} - J_{\mathcal{T}_3,R}{}^{(1)}, R^{-}_{\mathcal{T}_3}{}^{(1)})\right) \\ &= \mathcal{S}_{\mathsf{P+B}}(50, 265, 10) \end{aligned}$$

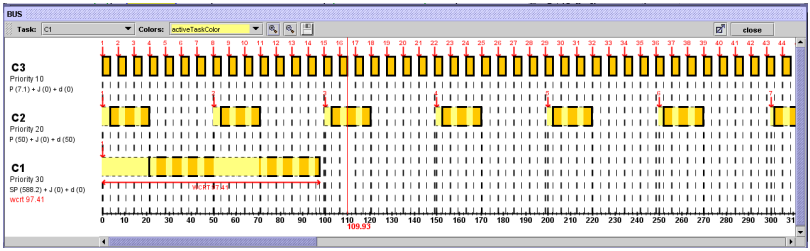



*Figure 7.9.* Bus Scheduling Diagram of First Analysis Cycle

Finally, we propagate the output stream of task $\mathcal{T}_3$ (step 5) to be considered in the second analysis cycle:

$$\mathcal{S}_{\mathcal{C}_2,\text{in}}{}^{(2)} = \mathcal{S}_{\mathcal{T}_3,\text{out}}{}^{(1)} = \mathcal{S}_{\mathsf{P+B}}(50, 265, 10)$$

### 7.6.2.2 Bus Analysis

These updated values from CPU analysis are not yet considered in the first cycle. Rather, the input streams of the three channels are still determined by the initial starting point assumptions:

$$\begin{aligned}
\mathcal{S}_{\mathcal{C}_1,\text{in}}{}^{(1)} &= \mathcal{S}_{\mathcal{C}_1,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{S}}(588.2) \\
\mathcal{S}_{\mathcal{C}_2,\text{in}}{}^{(1)} &= \mathcal{S}_{\mathcal{C}_2,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{P}}(50) \\
\mathcal{S}_{\mathcal{C}_3,\text{in}}{}^{(1)} &= \mathcal{S}_{\mathcal{C}_3,\text{in}}{}^{(0)} = \mathcal{S}_{\mathsf{P}}(7.14)
\end{aligned}$$

The response times are:

$$R^{+}_{\mathcal{C}_3}{}^{(1)} = 4.3 \qquad R^{-}_{\mathcal{C}_3}{}^{(1)} = 3.43$$

$$R^{+}_{\mathcal{C}_2}{}^{(1)} = 25.31 \qquad R^{-}_{\mathcal{C}_2}{}^{(1)} = 17.58$$

$$R^{+}_{\mathcal{C}_1}{}^{(1)} = 97.41 \qquad R^{-}_{\mathcal{C}_1}{}^{(1)} = 72.97$$

The scheduling diagram of the critical instance of channel $\mathcal{C}_1$ in Figure 7.9 shows that there is no self-interference, and the communication backlog of all channels is one:

$$\mathit{backlog}_{\mathcal{C}_1}{}^{(1)} = \mathit{backlog}_{\mathcal{C}_2}{}^{(1)} = \mathit{backlog}_{\mathcal{C}_3}{}^{(1)} = 1$$

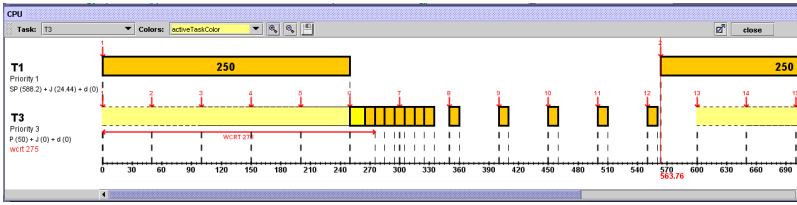



*Figure 7.10.* CPU Scheduling Diagram of Second Analysis Cycle

We combine steps 4 and 5 and obtain:

$$\begin{aligned}
\mathcal{S}_{\mathsf{DSP,in}}{}^{(2)} = \mathcal{S}_{\mathcal{C}_3,\mathsf{out}}{}^{(1)} &= \mathcal{S}_{\mathsf{P+J}}(T_{\mathcal{C}_3,\mathsf{in}}{}^{(1)}, J_{\mathcal{C}_3,R}{}^{(1)}) \\
&= \mathcal{S}_{\mathsf{P+J}}(7.14, 0.87) \\
\mathcal{S}_{\mathsf{HW,in}}{}^{(2)} = \mathcal{S}_{\mathcal{C}_2,\mathsf{out}}{}^{(1)} &= \mathcal{S}_{\mathsf{P+J}}(T_{\mathcal{C}_2,\mathsf{in}}{}^{(1)}, J_{\mathcal{C}_2,R}{}^{(1)}) \\
&= \mathcal{S}_{\mathsf{P+J}}(50, 7.73) \\
\mathcal{S}_{\mathcal{T}_1,\mathsf{in}}{}^{(2)} = \mathcal{S}_{\mathcal{C}_1,\mathsf{out}}{}^{(1)} &= \mathcal{S}_{\mathsf{S+J}}(T_{\mathcal{C}_1,\mathsf{in}}{}^{(1)}, J_{\mathcal{C}_1,R}{}^{(1)}) \\
&= \mathcal{S}_{\mathsf{S+J}}(588.2, 24.44)
\end{aligned}$$

## 7.6.3 Analysis Cycle 2

### 7.6.3.1 CPU Analysis

While task $\mathcal{T}_3$ has a constant timer-generated input, we now use the output stream of channel $\mathcal{C}_1$ from the first analysis cycle as the input stream of the second cycle for task $\mathcal{T}_1$:

$$\begin{aligned}
\mathcal{S}_{\mathcal{T}_1,\mathsf{in}}{}^{(2)} &= \mathcal{S}_{\mathcal{C}_1,\mathsf{out}}{}^{(1)} = \mathcal{S}_{\mathsf{S+J}}(588.2, 24.44) \\
\mathcal{S}_{\mathcal{T}_3,\mathsf{in}}{}^{(2)} &= \mathcal{S}_{\mathsf{timer}} = \mathcal{S}_{\mathsf{P}}(50)
\end{aligned}$$

Figure 7.10 shows the CPU scheduling diagram of this second analysis cycle. Compared to Figure 7.8, we see that the second activation of task $\mathcal{T}_1$ arrives earlier than in the previous analysis cycle. The reason is the input jitter that has increased: while assumed zero in the first cycle (due to the starting point generation), it now inherits the scheduling effects of channel $\mathcal{C}_1$ on the bus that are a result of the first analysis cycle (Section 7.6.2.2). This, however, does not yet affect the critical instant of task $\mathcal{T}_3$, and we obtain the same response times and backlogs as in the first cycle:

$$R^{+}_{\mathcal{T}_1}{}^{(2)} = 265 \qquad R^{-}_{\mathcal{T}_1}{}^{(2)} = 250 \qquad \mathit{backlog}_{\mathcal{T}_1}{}^{(2)} = 1$$

$$R^{+}_{\mathcal{T}_3}{}^{(2)} = 275 \qquad R^{-}_{\mathcal{T}_3}{}^{(2)} = 10 \qquad \mathit{backlog}_{\mathcal{T}_3}{}^{(2)} = 6$$

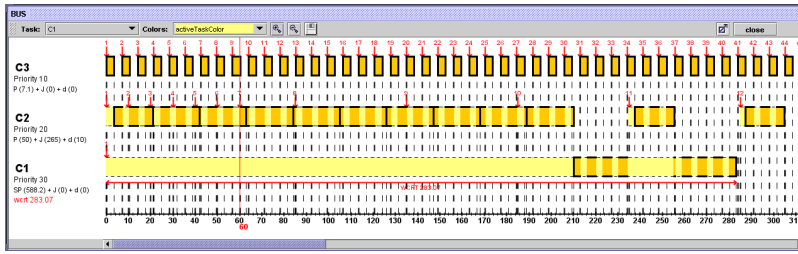



*Figure 7.11.* Bus Scheduling Diagram of Second Analysis Cycle

While these values remain constant, the output jitter of task $\mathcal{T}_1$ increases because it inherits the increased input jitter:

$$\begin{aligned} {\mathcal{S}_{\mathcal{T}_1,\text{out}}}^{(2)} &= \mathcal{S}_{\mathsf{S+J}}({T_{\mathcal{T}_1,\text{in}}}^{(2)}, {J_{\mathcal{T}_1,\text{in}}}^{(2)} + {J_{\mathcal{T}_1,R}}^{(2)}) \\ &= \mathcal{S}_{\mathsf{S+J}}(588.2, 39.44) \\ {\mathcal{S}_{\mathcal{T}_3,\text{out}}}^{(2)} = {\mathcal{S}_{\mathcal{C}_2,\text{in}}}^{(3)} &= \mathcal{S}_{\mathsf{P+B}} \left( \begin{array}{l} {T_{\mathcal{T}_3,\text{in}}}^{(2)}, \\ {J_{\mathcal{T}_3,R}}^{(2)}, \\ \max({T_{\mathcal{T}_3,\text{in}}}^{(2)} - q{J_{\mathcal{T}_3,R}}^{(2)}, {R^-_{\mathcal{T}_3}}^{(2)}) \end{array} \right) \\ &= \mathcal{S}_{\mathsf{P+B}}(50, 265, 10) \end{aligned}$$

### 7.6.3.2 Bus Analysis

Channels $\mathcal{C}_1$ and $\mathcal{C}_3$ are connected to environmental source tasks, so only the input of channel $\mathcal{C}_2$ is updated:

$$\begin{aligned} {\mathcal{S}_{\mathcal{C}_1,\text{in}}}^{(2)} &= {\mathcal{S}_{\mathcal{C}_1,\text{in}}}^{(0)} = \mathcal{S}_{\mathsf{S}}(588.2) \\ {\mathcal{S}_{\mathcal{C}_2,\text{in}}}^{(2)} &= {\mathcal{S}_{\mathcal{T}_3,\text{out}}}^{(1)} = \mathcal{S}_{\mathsf{P+B}}(50, 265, 10) \\ {\mathcal{S}_{\mathcal{C}_3,\text{in}}}^{(2)} &= {\mathcal{S}_{\mathcal{C}_3,\text{in}}}^{(0)} = \mathcal{S}_{\mathsf{P}}(7.14) \end{aligned}$$

The tremendous increase of the input jitter of channel $\mathcal{C}_2$ leads to communication bursts that can be seen in the scheduling diagram of Figure 7.11. We can see that the seventh event in that burst arrives (at $\Delta t = 60$) even before the third packet has been completely communicated, leading to long worst-case response times for both channels $\mathcal{C}_2$ and $\mathcal{C}_1$ and increasing communication backlogs. To analyze the burst, we have to use the windowing techniques mentioned in Section 7.5.2:

$${R^+_{\mathcal{C}_3}}^{(2)} = 4.3 \qquad {R^-_{\mathcal{C}_3}}^{(2)} = 3.43 \qquad {backlog_{\mathcal{C}_3}}^{(2)} = 1$$

$${R^+_{\mathcal{C}_2}}^{(2)} = 87.94 \qquad {R^-_{\mathcal{C}_2}}^{(2)} = 17.58 \qquad {backlog_{\mathcal{C}_2}}^{(2)} = 5$$

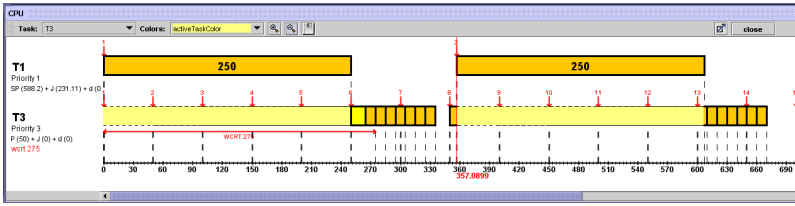



*Figure 7.12.* CPU Scheduling Diagram of Third Analysis Cycle

$$R^{+}_{\mathcal{C}_1}{}^{(2)} = 283.07 \qquad R^{-}_{\mathcal{C}_1}{}^{(2)} = 51.96 \qquad backlog_{\mathcal{C}_1}{}^{(2)} = 1$$

The propagated output streams are:

$$\begin{aligned}
\mathcal{S}_{\mathsf{DSP},\mathrm{in}}{}^{(3)} = \mathcal{S}_{\mathcal{C}_3,\mathrm{out}}{}^{(2)} &= \mathcal{S}_{\mathsf{P+J}}(T_{\mathcal{C}_3,\mathrm{in}}{}^{(2)}, J_{\mathcal{C}_3,R}{}^{(2)}) \\
&= \mathcal{S}_{\mathsf{P+J}}(7.14, 0.87) \\
\mathcal{S}_{\mathsf{HW},\mathrm{in}}{}^{(3)} = \mathcal{S}_{\mathcal{C}_2,\mathrm{out}}{}^{(2)} &= \mathcal{S}_{\mathsf{P+B}}(T_{\mathcal{C}_2,\mathrm{in}}{}^{(2)}, J_{\mathcal{C}_2,\mathrm{in}}{}^{(2)} + J_{\mathcal{C}_2,R}{}^{(2)}) \\
&= \mathcal{S}_{\mathsf{P+B}}(50, 335.36, 17.58) \\
\mathcal{S}_{\mathcal{T}_1,\mathrm{in}}{}^{(3)} = \mathcal{S}_{\mathcal{C}_1,\mathrm{out}}{}^{(2)} &= \mathcal{S}_{\mathsf{S+J}}(T_{\mathcal{C}_1,\mathrm{in}}{}^{(2)}, J_{\mathcal{C}_1,\mathrm{in}}{}^{(2)} + J_{\mathcal{C}_1,R}{}^{(2)}) \\
&= \mathcal{S}_{\mathsf{S+J}}(588.2, 231.11)
\end{aligned}$$

## 7.6.4 Analysis Cycle 3

### 7.6.4.1 CPU Analysis

The input jitter of $\mathcal{T}_1$ has further increased:

$$\begin{aligned}
\mathcal{S}_{\mathcal{T}_1,\mathrm{in}}{}^{(2)} &= \mathcal{S}_{\mathcal{C}_1,\mathrm{out}}{}^{(1)} = \mathcal{S}_{\mathsf{S+J}}(588.2, 231.11) \\
\mathcal{S}_{\mathcal{T}_3,\mathrm{in}}{}^{(2)} &= \mathcal{S}_{\mathsf{timer}} = \mathcal{S}_{\mathsf{P}}(50)
\end{aligned}$$

However, this has still no critical effect on the worst-case scheduling, illustrated in Figure 7.12. Again, we obtain the same response times and backlogs as in the previous cycles:

$$R^{+}_{\mathcal{T}_1}{}^{(3)} = 265 \qquad R^{-}_{\mathcal{T}_1}{}^{(3)} = 250 \qquad backlog_{\mathcal{T}_1}{}^{(3)} = 1$$

$$R^{+}_{\mathcal{T}_3}{}^{(3)} = 275 \qquad R^{-}_{\mathcal{T}_3}{}^{(3)} = 10 \qquad backlog_{\mathcal{T}_3}{}^{(3)} = 6$$

Only the output jitter of task $\mathcal{T}_1$ increases:

$$\begin{aligned}
{\mathcal{S}_{\mathcal{T}_1,\text{out}}}^{(3)} &= \mathcal{S}_{\mathsf{S+J}}({T_{\mathcal{T}_1,\text{in}}}^{(3)}, {J_{\mathcal{T}_1,\text{in}}}^{(3)} + {J_{\mathcal{T}_1,R}}^{(3)}) \\
&= \mathcal{S}_{\mathsf{S+J}}(588.2, 246.11) \\
{\mathcal{S}_{\mathcal{T}_3,\text{out}}}^{(3)} &= \mathcal{S}_{\mathsf{P+B}}\left({T_{\mathcal{T}_3,\text{in}}}^{(3)}, {J_{\mathcal{T}_3,R}}^{(3)}, \max({T_{\mathcal{T}_3,\text{in}}}^{(3)} - {J_{\mathcal{T}_3,R}}^{(3)}, {R^-_{\mathcal{T}_3}}^{(3)})\right) \\
&= \mathcal{S}_{\mathsf{P+B}}(50, 265, 10)
\end{aligned}$$

#### 7.6.4.2 Bus Analysis

When we look at the bus input streams, we see that these have not changed compared to the previous analysis cycle:

$$\begin{aligned}
{\mathcal{S}_{\mathcal{C}_1,\text{in}}}^{(3)} &= {\mathcal{S}_{\mathcal{C}_1,\text{in}}}^{(2)} = \mathcal{S}_{\mathsf{S}}(588.2) \\
{\mathcal{S}_{\mathcal{C}_2,\text{in}}}^{(3)} &= {\mathcal{S}_{\mathcal{C}_2,\text{in}}}^{(2)} = \mathcal{S}_{\mathcal{T}_3,\text{out}}{}^{(2)} = \mathcal{S}_{\mathsf{P+B}}(50, 265, 10) \\
{\mathcal{S}_{\mathcal{C}_3,\text{in}}}^{(3)} &= {\mathcal{S}_{\mathcal{C}_3,\text{in}}}^{(2)} = \mathcal{S}_{\mathsf{P}}(7.14)
\end{aligned}$$

Hence, the bus does not need to be analyzed in this cycle, nor do the output streams change:

$$\begin{aligned}
{\mathcal{S}_{\mathcal{C}_1,\text{out}}}^{(3)} &= {\mathcal{S}_{\mathcal{C}_1,\text{out}}}^{(2)} = \mathcal{S}_{\mathsf{S+J}}(588.2, 231.11) \\
{\mathcal{S}_{\mathcal{C}_2,\text{out}}}^{(3)} &= {\mathcal{S}_{\mathcal{C}_2,\text{out}}}^{(2)} = \mathcal{S}_{\mathsf{P+B}}(50, 335.36, 17.58) \\
{\mathcal{S}_{\mathcal{C}_3,\text{out}}}^{(3)} &= {\mathcal{S}_{\mathcal{C}_3,\text{out}}}^{(2)} = \mathcal{S}_{\mathsf{P+J}}(7.14, 0.87)
\end{aligned}$$

### 7.6.5 Analysis Cycle 4: Termination

Because the bus outputs have not changed in the previous cycle, the CPU inputs have also not changed:

$$\begin{aligned}
{\mathcal{S}_{\mathcal{T}_1,\text{in}}}^{(4)} &= {\mathcal{S}_{\mathcal{T}_1,\text{in}}}^{(3)} = {\mathcal{S}_{\mathcal{C}_1,\text{out}}}^{(3)} = \mathcal{S}_{\mathsf{S+J}}(588.2, 231.11) \\
{\mathcal{S}_{\mathcal{T}_3,\text{in}}}^{(4)} &= {\mathcal{S}_{\mathcal{T}_3,\text{in}}}^{(3)} = \mathcal{S}_{\mathsf{P}}(50)
\end{aligned}$$

Now, both components no longer change their input-output streams, and the entire analysis procedure has successfully terminated.

### 7.6.6 Sink Task Input Requirements

After the analysis of the *internal* system behavior, we finally check if the *external* behavior meets the requirements of the environmental sink tasks (see Section 7.5.1.4). In Section 7.2/Step 2, we have introduced the sub-steps for input stream capturing that possibly require event model interfacing and/or adaptation. The same concepts can be used to match the event model constraints imposed by environmental sink tasks.

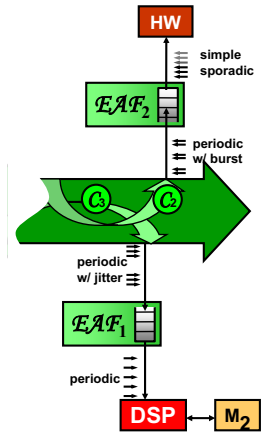



*Figure 7.13.* Required Event Adaptation Functions $\mathcal{EAF}$

We start with the DSP sub-system and see that the actual output stream from channel $\mathcal{C}_3$ does not meet the input requirement of the DSP:

$$\mathcal{S}_{\mathcal{C}_3,\text{out}} = \mathcal{S}_{\mathsf{P+J}}(7.14, 0.87) \notin \mathcal{R}_{\mathsf{DSP,in}} \subset \mathcal{EM}_{\mathsf{P}} \tag{7.8}$$

There is no plain event model interface for this model combination, and we must eliminate the jitter by means of an adaptation function. We apply *automatic shaping* from Section 6.5, and we obtain a periodic shaper with a buffer size of 1 and a maximum buffering delay of 8.01 time units. This is inserted as $\mathcal{EAF}_1$ between channel $\mathcal{C}_3$ and the DSP, illustrated in Figure 7.13. The actual input to the DSP is determined by Equation 6.53:

$$\begin{aligned} \mathcal{S}_{\mathsf{DSP,in}} &= \mathcal{EAF}_{\text{auto},1}(\mathcal{S}_{\mathcal{EAF}_1,\text{in}}, \mathcal{R}_{\mathsf{DSP,in}}) \\ &= \mathcal{EAF}_{\text{auto},1}(\mathcal{S}_{\mathcal{C}_3,\text{out}}, \{\mathcal{S}_{\mathsf{P}}(T_{\mathsf{DSP}} = T_{\mathsf{IP}} = 7.14)\}) \\ &= \mathcal{S}_{\mathsf{P}}(7.14) \in \mathcal{R}_{\mathsf{DSP,in}} \end{aligned} \tag{7.9}$$

The actual input stream of the HW component does not meet the requirement, either:

$$\mathcal{S}_{\mathcal{C}_2,\text{out}} = \mathcal{S}_{\mathsf{P+B}}(50, 335.36, 17.58) \notin \mathcal{R}_{\mathsf{HW,in}} \subset \mathcal{EM}_{\mathsf{S}} \tag{7.10}$$

For this situation, an event model interface exists, and we derive the transformed HW input stream according to Section 5.7.7. However, the sporadic period is too little and not supported by the input requirement:

$$\mathcal{EMIF}_{\mathsf{P+B}\rightarrow\mathsf{S}}(\mathcal{S}_{\mathcal{C}_2,\text{out}}) = \mathcal{S}_{\mathsf{S}}(T = 17.58) \notin \mathcal{R}_{\mathsf{HW,in}} = \{\mathcal{S}_{\mathsf{S}}(T_{\mathsf{HW}} = 20)\} \tag{7.11}$$

We finally apply automatic shaping and obtain the corresponding event adaptation function $\mathcal{EAF}_2$, a sporadic shaper with a backlog of 2 and a maximum delay of 24.2 time units. The actual HW input stream is:

$$\begin{aligned} \mathcal{S}_{\mathsf{HW,in}} &= \mathcal{EAF}_{\text{auto},2}(\mathcal{S}_{\mathcal{EAF}_2,\text{in}}, \mathcal{R}_{\mathsf{HM,in}}) \\ &= \mathcal{S}_{\mathsf{S}}(20) \in \mathcal{R}_{\mathsf{HW,in}} \end{aligned} \tag{7.12}$$

## 7.6.7 Results

We have analyzed the example with our SymTA/S tool suite [114] that implements all concepts presented in this thesis. It is written in Java and does not require any external analysis packages. All experiments that are presented in this thesis have been carried out on an Athlon1800+ (@1.15GHz) powered PC with 512 MByte of RAM, running the WindowsXP Professional operating system and the Java virtual machine from Sun Microsystem's Java2SDK, Standard Edition Version 1.4.1. The analysis run-time for the above example was below 1 second.

### 7.6.7.1 Overview

Table 7.1 provides a compact overview about the experiment. For each analysis cycle, the input and output jitters and the response times are shown. The analysis assumes zero input jitter for the first cycle. Each cycle yields updated values for the output jitters. The output jitter of channel $\mathcal{C}_1$ turns into the input jitter of task $\mathcal{T}_1$, and the output jitter of task $\mathcal{T}_3$ turns into the input jitter of channel $\mathcal{C}_2$. These values are shown in bold numbers. At the second analysis cycle, updated values need be considered for both elements in the dependency cycle, namely task $\mathcal{T}_3$ and channel $\mathcal{C}_2$. For the third cycle, the input jitter of task $\mathcal{T}_1$ has further increased, while the input jitter of channel $\mathcal{C}_2$ has not. The large jitters lead to self-interference, and consequently burst communication on channel $\mathcal{C}_2$. The rightmost column shows the size of the busy window [69, 121] required to analyze the bursts.

### 7.6.7.2 Jitter Observations

The monotonicity of jitters and response times play a key role in the definitions of a suitable starting point as well as a reasonable termination condition for the iterative analysis procedure. The example illustrates these effects. Jitters increase from inputs to outputs, and we have also seen the cyclic dependencies in detail. Figure 7.14 allows comparison of the changes in the worst-case scheduling scenarios from one analysis cycle to the next. The particular effect of increasing jitters is highlighted. On the CPU, we see that the task $\mathcal{T}_3$ is blocked for several activations and then runs in burst mode. The rightmost column of Table 7.1 indicates that the size of the busy window [69, 121] is seven (7). Even though the increasing input jitter of $\mathcal{T}_1$ leads to earlier re-arrivals (see

| task | input jitter $[\mu s]$ | resp. time $[\mu s]$ | output jitter $[\mu s]$ | size of busy window |
|---|---|---|---|---|
| Cycle 1 | | | | |
| $\mathcal{C}_1$ | 0 | [72.97 ; 97.41] | **24.44** | 1 |
| $\mathcal{C}_2$ | **0** | [17.58 ; 25.31] | 7.73 | 1 |
| $\mathcal{C}_3$ | 0 | [3.43 ; 4.3] | 0.87 | 1 |
| $\mathcal{T}_1$ | **0** | [250 ; 265] | 15 | 1 |
| $\mathcal{T}_3$ | 0 | [10 ; 275] | **265** | 7 |
| Cycle 2 | | | | |
| $\mathcal{C}_1$ | 0 | [51.96 ; 283.07] | **231.11** | 1 |
| $\mathcal{C}_2$ | **265** | [17.58 ; 87.94] | 335.36 | 10 |
| $\mathcal{C}_3$ | 0 | [3.43 ; 4.3] | 0.87 | 1 |
| $\mathcal{T}_1$ | **24.44** | [250 ; 265] | 39.44 | 1 |
| $\mathcal{T}_3$ | 0 | [10 ; 275] | **265** | 7 |
| Cycle 3 | | | | |
| $\mathcal{C}_1$ | 0 | [51.96 ; 283.07] | **231.11** | 1 |
| $\mathcal{C}_2$ | **265** | [17.58 ; 87.94] | 335.36 | 10 |
| $\mathcal{C}_3$ | 0 | [3.43 ; 4.3] | 0.87 | 1 |
| $\mathcal{T}_1$ | **231.11** | [250 ; 265] | 246.11 | 1 |
| $\mathcal{T}_3$ | 0 | [10 ; 275] | **265** | 7 |
| Termination after third analysis cycle!!! | | | | |

*Table 7.1.* Experiment Results

Figure 7.14), these still do not influence the critical instant of task $\mathcal{T}_3$ whose scheduling remains constant. On the bus, the burst output of task $\mathcal{T}_3$ leads to communication bursts that are only limited by the minimum event distance $\delta^-_{\mathcal{C}_2,\text{in}}(2) = d_{\mathcal{T}_3,\text{out}} = 10$. The size of the corresponding busy window is ten (10).

#### 7.6.7.3 Path Latencies

According to Definition 7.2, we define three paths in the example system. Path $\mathcal{P}_1$ from the output of the IP sub-system to the input of the DSP, $\mathcal{P}_2$ from the RTOS timer to the HW sub-system, and a third path from the sensor to the output (completion) of task $\mathcal{T}_1$:

$$\begin{aligned} \mathcal{P}_1 &= \{\mathcal{C}_3, \mathcal{EAF}_1\} \\ \mathcal{P}_2 &= \{\mathcal{T}_3, \mathcal{C}_2, \mathcal{EAF}_2\} \\ \mathcal{P}_3 &= \{\mathcal{C}_1, \mathcal{T}_1\} \end{aligned}$$

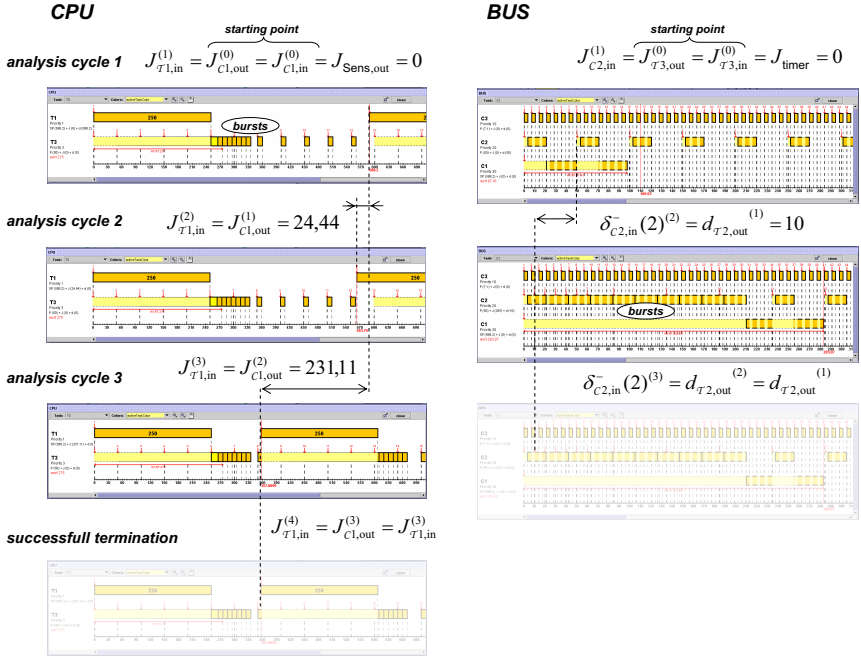



*Figure 7.14.* Scheduling Diagrams of All Analysis Cycles

According to Definition 7.3, the maximum latencies of these paths are:

$$\begin{aligned} \mathcal{PL}_1^+ &= R^+_{\mathcal{C}_3} + \mathit{delay}^+_{\mathcal{EAF}_1} \\ &= 4.3 + 8.01 = 12.31 \qquad (7.13) \\ \mathcal{PL}_2^+ &= R^+_{\mathcal{T}_3} + R^+_{\mathcal{C}_2} + \mathit{delay}^+_{\mathcal{EAF}_2} \\ &= 275 + 87.94 + 24.2 = 387.14 \qquad (7.14) \\ \mathcal{PL}_3^+ &= R^+_{\mathcal{C}_1} + R^+_{\mathcal{T}_1} \\ &= 283.07 + 265 = 548.07 \qquad (7.15) \end{aligned}$$

Finally, we calculate the accumulated backlog along each path:

$$\begin{aligned} backlog^+_{\mathcal{P}_1} &= backlog^+_{\mathcal{C}_3} + backlog^+_{\mathcal{EAF}_1} \\ &= 1 + 2 = 3 \qquad (7.16) \\ backlog^+_{\mathcal{P}_2} &= backlog^+_{\mathcal{T}_3} + backlog^+_{\mathcal{C}_2} + backlog^+_{\mathcal{EAF}_2} \\ &= 6 + 5 + 2 = 13 \qquad (7.17) \\ backlog^+_{\mathcal{P}_3} &= backlog^+_{\mathcal{C}_1} + backlog^+_{\mathcal{T}_1} \\ &= 1 + 1 = 2 \qquad (7.18) \end{aligned}$$

## 7.7 Optimizations

We have shown in Section 6.4.3 that shaping supports us in reducing transient load peaks. We have performed a set of experiments with periodic and sporadic shapers with different time-out values in order to optimize a *local schedule*. Shaping also has, however, an important global effect since the peak load reduction and load balancing is not limited to a single task or resource. A more deterministic execution or communication usually results in more deterministic output streams, in turn improving the scheduling of the next task or channel. This is of particular importance for systems with cycles and, as we shall see, it has a large impact on the scheduling dynamics as well as the analysis results.

### 7.7.1 Full Re-Synchronization

We insert a shaper $\mathcal{EAF}_3$ between task $\mathcal{T}_3$ and channel $\mathcal{C}_2$ thereby reducing the communication bursts on the bus. We have already indicated the possibility of shaping in Figure 7.6. We start with a fully periodic shaper to re-synchronize the output of task $\mathcal{T}_3$ to its original period. In other words, we eliminate the jitter. The overall analysis procedure does not change, but we have to consider the influence of shaping during the event stream propagation step (step 5 in Figure 7.2).

As we use the same analysis starting point, the first analysis cycle is mostly identical to the situation without shaping (see Section 7.6.2 for details). From the output stream of task $\mathcal{T}_3$, we can then dimension the shaper according to Section 6.3.4:

$$backlog^+_{\mathcal{EAF}_3} = 3 \qquad delay^+_{\mathcal{EAF}_3} = 315$$

Now, because of the shaper, the bus inputs do not change compared to the initial starting point. That means that the bus does not need to be analyzed a second time, since the periodic shaper breaks up the dependency cycle. Only the CPU must be analyzed again to account for the influence of bus scheduling. We can re-use the calculations from Section 7.6.3.1, and we see that the output stream of task $\mathcal{T}_3$ does not change compared to the first cycle, and the

| Shaper | Maximum Path Delay | | | Maximum Path Backlog | | | Analysis | Run- |
|---|---|---|---|---|---|---|---|---|
| | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ | Cycles | Time |
| — | 12.31 | 387.14 | 548.07 | 3 | 13 | 2 | 4 | <1s |
| P | 12.31 | 615.31 | 362.41 | 3 | 14 | 2 | 2 | <1s |
| S(30) | 12.31 | 424.31 | 443.02 | 3 | 12 | 2 | 4 | <1s |

*Table 7.2.* Path Properties of All Experiments

analysis terminates. For the local analysis of the CPU and the bus, the results of Sections 7.6.2.2 and 7.6.3.1 apply. In order to determine the path latencies and backlogs, we have to consider the shaper $\mathcal{EAF}_3$ in the path $\mathcal{P}_2$:

$$\mathcal{P}_2 = \{\mathcal{T}_3, \mathcal{EAF}_3, \mathcal{C}_2, \mathcal{EAF}_2\}$$

Table 7.2 summarizes the path properties of this example system with different shaper set-ups. The first two rows of the table allow the un-shaped version ("–") to be compared to the shaped version ("P"). The additional buffering has lead to a larger backlog and delay for path $\mathcal{P}_2$. However, the delay of path $\mathcal{P}_3$ is remarkably less than without the additional shaper, since the interference that channel $\mathcal{C}_1$ experiences has been reduced. Path $\mathcal{P}_1$ has not changed because the channel $\mathcal{C}_3$ that contributes to that path has the highest priority on the bus and is therefore not affected by other communications.

There is another advantageous side effect of shaping. Because of the shaper, the communication on channel $\mathcal{C}_2$ experiences less interference, and its output jitter is considerably less compared to the un-shaped version:

$$J_{\mathcal{C}_2,\text{out}} = 7.73 < 335.36 = J_{\mathcal{C}_2,\text{out,un-shaped}} \tag{7.19}$$

In effect, the minimum output event distance $\delta^-(2)$ in that stream is increasing from 17.58 (see Equation 7.11) to

$$\delta^-_{\mathcal{C}_2,\text{out}} = \max(T_{\mathcal{C}_2,\text{out}} - J_{\mathcal{C}_2,\text{out}}, R^-_{\mathcal{C}_2}) = 50 - 7.73 = 42.27$$

That means, we can find a plain $\mathcal{EMIF}$ such that the HW sub-system requirement is automatically met *without* the need for shaper $\mathcal{EAF}_2$:

$$\mathcal{EMIF}_{\mathsf{P+B}\rightarrow\mathsf{S}}(\mathcal{S}_{\mathcal{C}_2,\text{out}}) = \mathcal{S}_{\mathsf{S}}(T = 42.27) \in \mathcal{R}_{\mathsf{HW},\text{in}} = \{\mathcal{S}_{\mathsf{S}}(T_{\mathsf{HW}} = 20)\} \tag{7.20}$$

This demonstrates the large impact of buffering. A shaper is required on either position but the choice can be subject to further optimization.

### 7.7.2 Dynamic Bus Load Reduction

We now replace the fully periodic shaper with a sporadic shaper. We choose a time-out value of 30 which is in the "medium range" between the maximum allowed value (the period: 50) and the minimum reasonable value (the minimum distance: 10). The shaper also reduces the amount of interference on

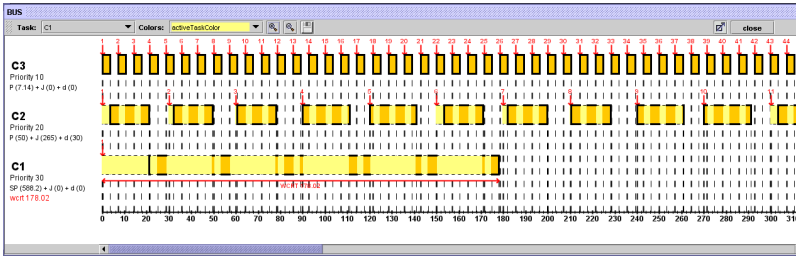



*Figure 7.15.* Bus Scheduling Diagram with Sporadic Shaper

the bus. We have to perform the shaper dimensioning of Section 6.4.1 in each analysis cycle. The analysis terminates in the fourth cycle. The worst-case communication scheduling scenario is shown in Figure 7.15, the path latency and backlog is provided in the last row of Table 7.2. We see that this option yields the optimal path backlog while the path delays are within the range of the other two experiments. Again, the shaper $\mathcal{EAF}_2$ becomes obsolete because the output distance if channel $\mathcal{C}_2$ is

$$\delta^-_{\mathcal{C}_2,\text{out}} = \max(\underbrace{T_{\mathcal{C}_2,\text{out}} - J_{\mathcal{C}_2,\text{out}}}_{<0}, \underbrace{\delta^-_{\mathcal{C}_2,\text{in}} - J_{R_{\mathcal{C}_2}}}_{30-7.73=22.73}, \underbrace{R^-_{\mathcal{C}_2}}_{17.58}) = 22.73$$

which fulfills the HW input requirement.

### 7.7.3 Reducing the Bus Speed

The examples show that the proposed analysis procedure yields detailed information about the timing and performance of each individual task or communication, as well as resource utilization information. For instance, the bus has currently a utilization of 74,3% (see Equation 7.7). In a final set of experiments, we reduce the bus speed to 80% of its original speed, i. e. 240Mbyte/s. This increases the individual byte and packet communication times, and the resulting overall network utilization is:

$$\mathcal{U}_{\text{net}} = \frac{U_{\text{net}}}{240 Mbyte/s} = 92,8\% \tag{7.21}$$

We perform four experiments with different set-ups for shaper $\mathcal{EAF}_3$. Again, the first analysis cycle of all experiments is identical. We perform the first experiment without an additional shaper. We can observe that the path delays on paths $\mathcal{P}_2$ and $\mathcal{P}_3$ –along with the jitters and response times– rapidly increase. They exceed their absolute deadlines, which are set to 2000 and 2500, after the sixth analysis cycle, and the analysis terminates without success. The overall analysis run-time of this experiment was below two seconds. The rapid

| | Maximum Path Delay | | | Maximum Path Backlog | | | Analysis | Run- |
|---|---|---|---|---|---|---|---|---|
| Shaper | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ | $\mathcal{P}_1$ | $\mathcal{P}_2$ | $\mathcal{P}_3$ | Cycles | Time |
| — | 13.56 | $>2000$ | $>2500$ | 3 | $>46$ | $>8$ | 6! | $<$2s |
| P | 13.56 | 630.21 | 560.41 | 3 | 14 | 2 | 2 | $<$1s |
| S(50) | 13.56 | 630.21 | 560.41 | 3 | 14 | 2 | 2 | $<$1s |
| S(40) | 13.56 | 950.21 | 1132.6 | 3 | 23 | 4 | 5 | $<$1s |

*Table 7.3.* Path Properties of All Experiments with Reduced Bus Speed

increase of the response times demonstrates the usefulness of the termination condition of Section 7.4.3 in practice.

In the second experiment, we add a periodic shaper, and the analysis successfully terminates after two cycles. Next, we apply maximum sporadic shaping with a time-out of 50. Finally, we reduce the time-out value to 40. The results are presented in Table 7.3. All experiments with shapers terminate within the first second. We see that both the periodic shaper and the maximum sporadic shaper yield identical path delays and backlogs. They differ, however, in the scheduling of channel $\mathcal{C}_1$ on the bus which leads to a higher activation jitter of task $\mathcal{T}_1$. Neither of the effects is illustrated further here.

### 7.7.4 Concluding Remarks

We have used this example system earlier at the IEEE Real-Time Systems Symposium, 2003, in Cancun, Mexico. At the time we wrote the paper [98], we could not use the SymTA/S tool, so we had to do the calculations manually. Unfortunately, we made a systematic mistake by not appropriately considering the input jitters, and the results published in [98] have flaws. With this paragraph, we would like to apologize for this mistake and kindly ask this to be considered when comparing the results.

## 7.8 Summary

In this Chapter, we have presented our novel system-level scheduling analysis procedure for heterogeneous distributed systems. Using an expressive example, we have shown in detail how the individual concepts of the preceding chapters are applied one after another according to a systematically structured seven-step analysis procedure. In practice, each individual step is comprehensible and can be performed without considering any specialties of the model, the architecture, or individual parts of it. Event stream incompatibilities are identified and resolved step-by-step. The same holds for the way the system-level dependencies are captured and analyzed. We have seen that even critical design pitfalls, such as dependency cycles, do not increase the complexity of the overall procedure at all. On the contrary, the structured, cyclic approach provides meaningful intermediate results that can already give hints for up-

coming bottlenecks. The actual amount of jitter increase, for instance, gives valuable feedback about the dynamic evolution of the analysis.

From the mathematical perspective, we have shown that known deadlines and other constraints allow the algorithmic complexity of the analysis to be efficiently bounded, in case the analysis does not converge successfully. Finally, we have shown that, and how, traffic shaping can substantially reduce the analysis complexity, enforce convergence, and optimize the system globally.

The experiments illustrate the main contribution of this thesis. The event stream representation and the structured standard event models are the key enablers for both, the novel analysis procedure to determine the timing and the performance of the system, and the comprehensive view that increases system understanding and provides efficient and extremely fast optimization support.

Chapter 8

# SUMMARY AND CONCLUSION

## 8.1 Summary

In order to find a reasonable formal basis for our technique, we reviewed popular scheduling analysis techniques and landmark publications. We discovered that the host of analysis proposals share only few relatively compact and comprehensible event models for task activation modeling. Looking for a way to combine these techniques into a system-level analysis, we thoroughly investigated the specific properties and the characteristic functions of the four most popular *input event models*, as well as the influence of scheduling with respect to *output event models* that have, interestingly, been scarcely considered to date. We showed that the existing input models have weaknesses in output stream modeling. The concept of increasing jitter that leads to event bursts has not previously been systematically analyzed. The same applies to the close relationship between periodic and sporadic models.

Based on these observations, we structured input and output models into two appropriate classes: periodic and sporadic, and we have three models in each class: strict, jitter, and burst. This systematic arrangement leads to a self-contained six-class model that represents an efficient compromise between model simplicity, theoretical completeness, and practical applicability. In order to allow the heterogeneous use of these six models in one system, we defined compatibility and coverage tests for event streams and models, and we subsequently developed *Event Model Interfaces* and *Event Adaptation Functions*. These provide the necessary model transformations that enable the integration of different components, and the composition of their local analysis techniques within the novel system-level analysis procedure.

The proposed procedure propagates the outputs of one component to the inputs of the connected components, thereby analyzing the components locally,

one after another. The structured system model, and the use of simple, established event models, allows a comprehensive overview of all components and their interactions. The analysis provides key information about the local subsystem timing such as task response times and resource utilization, as well as global system properties such as end-to-end path latencies and overall buffering requirements. These are the key system parameters that designers and system architects require to verify the performance of their systems and components, and to evaluate the integration decisions. The approach presented is both comprehensible and provides this information faster than many previous attempts, including the best-practice of performance simulation.

We have investigated the impact of event stream dependency cycles that can turn the iterative analysis procedure into a convergence problem for event streams. We could show that the standard event models reveal monotonicity properties, that lead to a bounded number of analysis iterations until the analysis converges or a path constraint or a deadline is missed, which makes the analysis procedure applicable in practice; and even in the case of unsuccessful termination, the analysis provides additional information about which deadline is missed, and the stream representation provides initial hints to source route such problems.

## 8.2 Extensibility

The modularity of the analysis model allows not only to re-use existing approaches such as RMS [73], it also eases the development of new analysis modules for specific components. We have already characterized the popular automotive ERCOSek operating system [32], and we have developed a SymTA/S library component for that [16, 61]. Even though ERCOSek defines a sophisticated priority structure that complicates the local analysis, we could develop and implemented the module into out SymTA/S tool within very short time. Currently, we are investigating CAN in a project with a major automotive supplier. This modular extensibility is considered a major advantage over the holistic analysis approach, that suffers from the complexity to build the holistic models.

However, the *basic* SymTA/S approach, as far as introduced in this thesis, has certain obvious limitations. We have assumed that each task has exactly one input and one output. The input of one task is connected to exactly one output of another task, while one output can possibly be connected to several inputs. Consequently, the supported application topology is restricted to task trees, although the examples in this thesis consist of task chains, a subclass of trees. We have further assumed that each task is activated by a single input event, and the task produces at most one output event per execution. Realistic applications usually exhibit more complex task input-output behavior, more

complex activation conditions, more complex task topologies, and finally more complex system-level interactions.

The basic SymTA/S approach, however, represents a *core technology* that lays the foundation for a set of extensions. In order to deal with *data-rate transitions*, which are commonly found in signal processing applications including SDF systems, task execution and communication have been recently separated from task activation and scheduling. *Token production and consumption curves* have been introduced [54] to be associated with task inputs and outputs. Input token curves are transformed into activation event streams that are further used for task scheduling analysis. Likewise, output event streams are re-transformed into token curves to be propagated during the system level analysis. These transformations also consider *multi-input activation* with AND or OR semantics [58], which allows the scope of system-level event streams, and the system-level analysis procedure, to be extended to complex token-flow models and sophisticated application topologies including *functional cycles* [58].

These extensions are only possible because of the clear separation of local analysis and global interactions, a key contribution of this thesis. Moreover, it could be shown that the six-class event model set can efficiently capture both task activation and completion, as well as token consumption and production [54]. Furthermore, multi-input task activation exhibits backlogs and requires the insertion of appropriate buffers. In order to dimension the buffers and determine the additional delay, key conceptual contributions of this thesis can be re-used, that are related to event stream adaptation.

We summarize that the ideas presented in this thesis are *not generally limited* to simple applications. On the contrary, the basic approach provides a vehicle for a variety of extensions that benefit from: a) the clear separation of local analysis and global interaction modeling using efficient event models and b) from the idea of model transformations using appropriate interfaces and adaptations. In other words, *SymTA/S is the enabling technology* for the analysis of complex embedded applications that are implemented on heterogeneous hardware/software architectures. The details of the extensions mentioned can be found in Jersak's thesis [52].

Another type of extension benefits from the efficiency of the analysis procedure. We have seen in the experiments that the analysis run-time is in the range of seconds. This enables the analysis of a large number of different system setups in a relatively short period of time. The SymTA/S tool [114] has been coupled with an *automatic exploration and optimization* framework [13, 12]. The exploration framework uses evolutionary algorithms [147] to modify shaping and other implementation decisions, such as priorities or time slots, to find optimized solutions [44, 43]. Such optimizations are only possible with a sufficiently fast analysis procedure which SymTA/S provides.

A related research direction targets *sensitivity analysis*, where binary search trees [89] are used to determine the system reserves and describe the system flexibility during the design phase, whilst guaranteeing that system constraints can be satisfied. Again, the efficiency of the new approach enables the application of such kind of sensitivity analysis.

## 8.3 Outlook and Future Work

The conservative nature of the analysis that solely considers best-case and worst-case bounds is often a cause of concern. Clearly, the system-level analysis cannot be more accurate than the individual local models and techniques. In order to derive safe, guaranteed bounds, the parameter intervals can become large. Under certain circumstances, this may lead to conservative results. The holistic approaches take into account more detailed correlations between individual parameters and tasks, that are treated independently in our approach to reasons of modularity and flexibility. There are two distinct and apparently reasonable approaches to increasing the accuracy of the analysis.

By incorporating additional information about the task execution and communication behavior, we can distinguish between several task modes or contexts. *Intra-stream contexts* [48, 55] can be associated with event streams to model the correlated execution of tasks including their communication. It was shown that this considerably tightens analysis bounds for specific applications. Likewise, *inter-stream contexts* [48, 55] allow phase and offset information to be specified for task scheduling, consequently tightening response time analysis.

As the second alternative, one could investigate how much of the key holistic ideas can be incorporated into in the compositional SymTA/S approach to increase the accuracy without degrading modularity and efficiency. We specifically expect that the influence of bursts on path latencies can be more efficiently handled [35].

## 8.4 Conclusion

In this thesis we have presented a novel approach to analyzing the timing and the performance of heterogeneous, distributed real-time systems. One key contribution of this work is the comprehensible representation of the interactions between the individual components and sub-systems, which are a major source of complexity and make the timing analysis problem a huge challenge in practice.

Previous work in this field either –in case of *holistic analysis*– does not provide the necessary modularity and flexibility that today's "cut&paste" design style requires, or –in the case of the two known *compositional* approaches– requires relatively complex interaction models, that must be homogeneously

used in the entire system and substantially hinder the integration and re-use of known, established sub-system techniques.

The new approach is aimed explicitly at supporting the heterogeneous application of several different event models in one system. This flexibility allows a variety of known techniques for individual components and sub-systems to be used locally, without compromising global analysis, which we consider a key advantage with respect to the practical application of the approach.

# Bibliography


[1] *Accelera*. http://www.accellera.org.

[2] Heinz Arnold. Thema der woche (topic of the week): Automotive electronics, in German. *Markt & Technik*, (36):16–20, 2004.

[3] N. C. Audsley and A. Burns. Real-time system scheduling. Technical report, Department of Computer Science, University of York, England, 1990.

[4] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Journal of Real-Time Systems*, 8(5):284–292, 1993.

[5] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems*, pages 133–137, 1991.

[6] N. C. Audsley, K.W. Tindell, and A. Burns. The end of the line for static cyclic scheduling? In *In Proc. Euromicro Conference on Real-Time Systems*, 1993.

[7] AUTOSAR Development Partnership. *AUTomotive Open System Architecture – AUTOSAR*. http://www.autosar.org/.

[8] AXYS Design. *MaxSim Development Suite*. http://www.axysdesign.com/products/products_maxsim.asp.

[9] S. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman Hall, CRC Press, 2004.

[10] S. Baruah, L.E. Rosier, and R.R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real-Time Systems*, 2:301–324, 1990.

[11] J. Blazewicz. *Modeling and Performance Evaluation of Computer Systems*, chapter Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines. North-Holland, Amsterdam, 1976.

[12] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA – A platform and programming language independent interface for search algorithms. In *Evolutionary Multi-Criterion Optimization*, pages 494 – 508, 2003.

[13] T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. *Journal on Design Automation for Embedded Systems*, 3(8):23–58, January 1998.

[14] J. L. Boudec and P. Thiran. Network calculus - a theory of deterministic queuing systems for the internet. In *In Proceedings of International Workshop on Workshop on Quality in Multiservice IP Networks*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[15] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, 1998.

[16] J.-C. Braam. Analyse der Architektureinflüsse auf das Ausführungszeitverhalten von Software-Funktionen am Beispiel ERCOS$^{EK}$. Master's thesis, Technical University of Braunschweig, March 2003.

[17] G. Buttazzo. *HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, Massachusetts, USA, 1997.

[18] G. Buttazzo. Rate monotonic vs. edf: Judgment day. In *Proc. 3rd Workshop on Embedded Software (EMSOFT)*, Philadelphia (PA), USA, October 2003.

[19] Cadence. *Cierto VCC Environment*. http://www.cadence.com/products/vcc.html.

[20] Beatriz Asensio Calvo. Real-time analysis of time-driven scheduling – a quantitative comparison of Round Robin and TDMA. Technical report, Technical University of Braunschweig, September 2001.

[21] S. Chakraborty. *System-Level Timing Analysis and Scheuling for Embedded Packet Processors*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2003.

[22] S. Chakraborty, T. Erlenbach, S. Künzli, and L. Thiele. Schedulability of event-driven code blocks in real-time embedded systems. In *Proceedings Design Automation Conference*, pages 616–621, New Orleans, LA, USA, 2002.

[23] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. In *Proceedings IEEE Real-Time Systems Symposium*, Austin, TX, USA, 2002.

[24] M.-I. Chen and K.-J. Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *Journal of Real-Time Systems*, 2(4):325–346, 1990.

[25] CiA – Can in Automation. *TTCAN - Time-Triggered Controller Area Network*. http://www.can-cia.de/can/ttcan/.

[26] A. Colin and I. Puaut. Worst-case execution time analysis of the rtems real-time operating system. In *Proc. Euromicro Conference of Real-Time Systems*, pages 191–198, Delft, Nederlands, June 2001.

[27] R. Cruz. A calculus for network delay. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[28] A. Dasdan. *Timing Analysis of Embedded Real-Time Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[29] Petru Eles, Alexa Doboli, Paul Pop, and Zebo Peng. Scheduling with bus access optimization for distributed embedded systems. *IEEE Transactions on VLSI Systems*, 8(5):472–491, 2000.

[30] Petru Eles, Krzysztof Kuchcinski, Zebo Peng, Paul Pop, and Alexa Doboli. Scheduling of conditional process graphs for the synthesis of embedded systems. In *Proc. Design, Automation and Test in Europe - DATE*, 1998.

[31] M. Engels, G. Bilsen, R. Lauwereins, and J. Peperstraete. Cyclo-Static Data Flow: Model and implementation. In *Proc. 28th Asilomar Conf. on Signals, Systems, and Computers*, pages 503–507, Pacific Grove, CA, 1994.

[32] ETAS. *ERCOSek, OSEK based Real-Time Operating System*. http://www.etas.de/html/en/products/ec/ercosek/en_products_ec_ercosek_index.htm.

[33] ETAS, formerly Livedevices. *Real-Time Architect*. http://www.livedevices.co.uk/realtime.shtml.

[34] C. J. Fidge. Real-time schedulability tests for preemptive multitasking. *Journal of Real-Time Systems*, 14:61–93, 1998.

[35] M. Fidler. Extending the network calculus pay bursts only once principle to aggregate scheduling. In *In Proceedings of International Workshop on Quality of Service in Multiservice IP Networks*, Lecture Notes in Computer Science. Springer, 2003.

[36] FlexRay-Group. *FlexRay*. http://www.flexray.com.

[37] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[38] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123, Oulu, Finland, 1993.

[39] Klaus Gresser. *Echtzeitnachweis ereignisgesteuerter Realzeitsysteme*. PhD thesis, only avaliable in German, Technische Universität München, 1993.

[40] J. J. Gutierrez and M. G. Harbour. Increasing schedulability in distributed hard real-time systems. In *Proceedings 7th Euromicro Workshop on Real-Time Systems*, page 71, Odense, Denmark, June 1995.

[41] J. J. Gutierrez, J. C. Palencia, and M. G. Harbour. On the schedulability analysis for distributed hard real-time systems. In *Proceedings 9th Euromicro Workshop on Real-Time Systems*, pages 136–143, Toledo, Spain, June 1997.

[42] J. J. Gutierrez, J. C. Palencia, and M. G. Harbour. Schedulability analysis of distributed hard real-time systems with multiple-event synchronization. In *Proceedings 12th Euromicro Workshop on Real-Time Systems*, page 15, Stockholm, Sweden, June 2000.

[43] A. Hamann and R. Ernst. TDMA time slot and turn optimization with evolutionary search techniques. In *submitted to Design, Automation and Test in Europe Conference - DATE*, Munich, Germany, March 2005.

[44] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with SymTA/S - symbolic timing analysis for systems. In *Proceedings 25th International Real-Time Systems Symposium*, December 2004.

[45] M. G. Harbour, J. L. Medina, J. J. Gutierrez, J. C. Palencia, and J. M. Drake. Mast: An open environment for modeling, analysis, and design of real-time systems. In *1st Workshop on Computer Aided Architectural Analysis of Real-Time Systems*, Aranjuez, Spain, October 2002.

[46] C. Haubelt, J. Teich, and K. Richter. System design for flexibility. In *Proc. of Design, Automation and Test in Europe Conference (DATE'02)*, Paris, France, March 2002.

[47] W. Henderson, D. Kendall, and A. Robson. Improving the accuracy of scheduling analysis applied to distributed systems. *Journal of Time-Critical Computing Systems*, 20(1):5–25, 2001.

[48] R. Henia. Analyse von kontextabhängigem Systemverhalten komplexer eingebetteter Systeme. Master's thesis, Technical University of Braunschweig, April 2003.

[49] P.-E. Hladik and A.-M. Deplanche. Best-case response time analysis for precedence relations in hard real-time systems. In *Proc. Work-in-Progress Session, IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.

[50] IEEE. *The Official IEEE 802.5 Web Site*. http://www.ieee802.org/5/www8025org/.

[51] K. Jeffay. *The Real-Time Producer/Consumer Paradigm: Towards Verifiable Real-Time Computations*. PhD thesis, University of Washington, Department of Computer Science, 1989.

[52] M. Jersak. *Performance Analysis for Complex Embedded Applications*. PhD thesis, Technical University of Braunschweig, 2004.

[53] M. Jersak, Y. Cai, D. Ziegenbein, and R. Ernst. A transformational approach to constraint relaxation of a time-driven simulation model. In *Proceedings 13th International Symposium on System Synthesis*, Madrid, Spain, September 2000.

[54] M. Jersak and R. Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *Proceeding 40th Design Automation Conference*, Annaheim, USA, June 2003.

[55] M. Jersak, R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded system design. In *Proceeding Design Automation and Test in Europe*, Paris, France, March 2004.

[56] M. Jersak, R. Racu, J. Staschulat, K. Richter, R. Ernst, J.-C. Braam, and F. Wolf. Formal methods for integration of automotive software. In A.A. Jerraya, S. Yoo, D. Verkest, and N. Wehn, editors, *Embedded Software for SoC*, pages 11–24. Kluwer Academic Publishers, August 2003.

[57] M. Jersak, K. Richter, and R. Ernst. Combining complex event models and timing constraints. In *Proceedings 6th IEEE Workshop on High Level Design Validation and Test*, Monterey, USA, November 2001.

[58] M. Jersak, K. Richter, and R. Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, 2004.

[59] M. Jersak, K. Richter, R. Henia, R. Ernst, and F. Slomka. Transformation of SDL specifications for system-level timing analysis. In *Tenth International Symposium on Hardware/Software Codesign (CODES'02)*, Estes Park, Colorado, USA, May 2002.

[60] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[61] D. Kerk. Implementierung einer Scheduling-Analyse für das ERCOS$^{EK}$-Betriebssystem und Integration in SymTA/S. Master's thesis, Technical University of Braunschweig, May 2004.

[62] T. Kim, J. Lee, H. Shin, and N. Chang. Best case response time analysis for improved schedulability analysis of distributed real-time tasks. In *Proceedings ICDCS Workshops on Distributed Real-Time Systems*, pages 14–20, April 2000.

[63] M. Klein. *A Practitioner's Handbook for Real-Time Analysis*. Kluwer Academic Publishers, Boston, Massachusetts, USA, 1993.

[64] H. Kopetz and G. Gruensteidl. TTP - a time-triggered protocol for fault-tolerant computing. In *Proceedings 23rd International Symposium on Fault-Tolerant Computing*, pages 524–532, 1993.

[65] Hermann Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Massachusetts, april 1997.

[66] E. A. Lee. Recurrences, iteration, and conditionals in statically scheduled block diagram languages. In R. W. Brodersen and H. S. Morowitz, editors, *VLSI Signal Processing III*, pages 330–340, IEEE Press, New York, 1988.

[67] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[68] E. A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1), January 1987.

[69] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings Real-Time Systems Symposiom*, pages 201–209, 1990.

[70] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings Real-Time Systems Symposiom*, pages 166–171, IEEE Computer Society Press, 1989.

[71] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of peridic, real-time tasks. *Journal of Performance Evaluation*, 21(7):237–250, 1982.

[72] Lin Consortium. *LIN – Local Interconnect Network*. http://www.lin-subbus.org/.

[73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[74] J. Liu. *Real-Time Systems*. Prentice-Hall, Boston, Massachusetts, USA, 2000.

[75] C. D. Locke. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. *Journal of Real-Time Systems*, 4(1):37–52, 1992.

[76] A. Mathur, A. Dasdan, and R. K. Gupta. Rate Analysis for Embedded Systems. *ACM Transactions on Design Automation of Electronic Systems*, 3(3):408 – 436, July 1998.

[77] Mentor Graphics. *Seamless Co-Verification Environment*. http://www.mentorg.com/seamless/.

[78] A.K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[79] OAR Corporation. *RTEMS – Real-Time Executive for Multiprocessor Systems*. http://www.rtems.com.

[80] OSEK Steering Committee. *OSEK Open systems and the corresponding interfaces for automotive electronics*. http://www.osek-vdx.org/.

[81] J. C. Palencia, J. J. Gutierrez, and M. G. Harbour. Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems. In *Proceedings 10th Euromicro Workshop on Real-Time Systems*, page 35, Berlin, Germany, June 1998.

[82] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the IEEE Real-Time Systems Symposium*, page 26. IEEE Computer Society, 1998.

[83] J. C. Palencia and M. G. Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 328–399. IEEE Computer Society, 1999.

[84] J. C. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. In *Proceedings Euromicro Conference on Real-Time Systems*, July 2003.

[85] Philips Semiconductors. *Controller Area Network CAN*. http://www.semiconductors.philips.com/can/.

[86] P. Pop. *Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems*. PhD thesis, Linköping University, 2003.

[87] P. Pop, P. Eles, and Z. Peng. Bus access optimization for distributed embedded systems based on schedulability analysis. In *Proc. Design, Automation and Test in Europe (DATE 2000)*, Paris, France, 2000.

[88] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proc. Int. Symposium on Hardware/software codesign CODES'02*, Estes Park, USA, 2002.

[89] R. Racu, M. Jersak, and R. Ernst. Applying sensitivity analysis in real-time distributed systems. In *Submitted to 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'05)*, March 2005.

[90] R. Racu, K. Richter, and R. Ernst. Calculating task output event models to reduce distributed system cost. In *Proceedings of GI/ITG/GMM-Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, February 2004.

[91] R. Rajkumar, L. Sha, , and J. P. Lehoczky. Realtime synchronization protocols for multiprocessors. In *Proceedings Real-Time Systems Symposium*, pages 259–269, 1988.

[92] O. Redell and M. Sanfridson. Exact best-case response time analysis of fixed priority scheduled tasks. In *Proceedings 14th Euromicro Workshop on Real-Time Systems*, page 165, Vienna, Autria, June 2002.

[93] K. Richter. Developing a general model for scheduling of mixed transformative/reactive systems. Master's thesis, Technical University of Braunschweig, January 1998.

[94] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proc. of Design, Automation and Test in Europe Conference (DATE'02)*, Paris, France, March 2002.

[95] K. Richter and R. Ernst. A formal approach to performance verification of heterogeneous architectures. In *Proceedings Embedded World Conference*, Nürnberg, Germany, February 2004.

[96] K. Richter, R. Ernst, and W. Wolf. Hierarchical specification methods for platform-based design. In *Proc. of Tenth Workshop on Synthesis And System Integration of Mixed Technologies (SASIMI 2001)*, Nara, Japan, October 2001.

[97] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4), April 2003.

[98] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor SoC. In *Proceedings 24th International Real-Time Systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[99] K. Richter, D. Ziegenbein, R. Ernst, L. Thiele, and J. Teich. Representation of function variants for embedded system optimizetion and systhesis. In *Proceeding 36th Design Automation Conference*, pages 517–522, New Orleans, USA, June 1999.

[100] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Bottom-up performance analysis of HW/SW platforms. In *Proceedings Distributed and Parallel Embedded Systems Conference (DIPES'02)*, Montreal, Canada, August 2002.

[101] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proceeding 39th Design Automation Conference*, New Orleans, USA, June 2002.

[102] F. A. Schreiber. Is time a real time? An overview of time ontology in informatics. In W. A. Halang and A. D. Stoyenko, editors, *Real-Time Computing*. Springer Verlag, 1994.

[103] SDL Forum Society. *Specification and Description Language*. http://www.sdl-forum.org/SDL/index.htm.

[104] Semiconductor Industry Association. *2003 International Technology Roadmap for Semiconductors*. http://public.itrs.net/Files/2003ITRS/Home2003.htm.

[105] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[106] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, January 1994.

[107] Sonics. *SiliconBackplane $\mu$Network*. http://www.sonicsinc.com/Pages/Networks.html.

[108] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Journal of Real-Time Systems*, 1(1):27–60, 1989.

[109] M. Spuri. *Earliest Deadline Scheduled in Real-Time Systems*. PhD thesis, INRIA, Le Chesnay, Cedex, France, 1995.

[110] M. Spuri. Analysis of deadline scheduled real-time tasks. Technical report, INRIA, Le Chesnay, Cedex, France, 1996.

[111] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *DEADLINE SCHEDULING FOR REAL-TIME SYSTEMS – EDF and Related Algorithms*. Kluwer Academic Publishers, Boston, Massachusetts, USA, 1998.

[112] K. Strehl, L. Thiele, D. Ziegenbein, R. Ernst, and J. Teich. Scheduling hardware/software systems using symbolic techniques. In *Proc. CODES'99, the 7th Int. Workshop on Hardware/Software Co-Design*, pages 173–177, Rome, Italy, May 1999.

[113] A.S. Tanenbaum. *Operating Systems Design and Implementation*. Prentice-Hall, 1987.

[114] Technical University of Braunschweig. *SymTA/S – Symbolic Timing Analysis for Systems*. http://www.symta.org.

[115] The MathWorks. *Simulink datasheet*. http://www.mathworks.com/products/simulink/.

[116] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*, Geneva, Switzerland, 2000.

[117] L. Thiele, s. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. Embedded software in network processors - models and algorithms. In *Proc. 1st Workshop on Embedded Software (EMSOFT)*, Lake Tahoe (CA), USA, October 2001.

[118] L. Thiele, K. Strehl, D. Ziegenbein, R. Ernst, and J. Teich. FunState - an internal design representation for codesign. In *ICCAD'99, the IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 558–565, San Jose, U.S.A., November 1999.

[119] L. Thiele, J. Teich, and D. Ziegenbein. FunState - functions driven by state machines. Technical Report 33, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zürich, January 1998.

[120] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Department of Computer Science, University of York, UK, 1994.

[121] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, Mar 1994.

[122] K. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. *Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, 40:117–134, 1994.

[123] K. Tindell, J. Clark, and A. Wellings. Analysing real-time communications: Controller area network. In *Proceedings International Real-Time Systems Symposium*, pages 259–263, 1994.

[124] Tri-Pacific Software, Inc. *RAPID RMA*. http://www.tripac.com/html/prod-fact-rrm.html.

[125] TTTech AG. *TTP - Time-Triggered Protocol*. http://www.tttech.com/.

[126] TTTech AG. *TTP Software Development Suite*. http://www.tttech.com/products/software/ttptools/overview.htm.

[127] University of California at Berkeley. *Giotto – A Methodology for Embedded Control Systems Development*. http://www-cad.eecs.berkeley.edu/ fresco/giotto/.

[128] University of Cantabria, Spain. *MAST – Modeling and Analysis Suite for Real-Time Applications*. http://mast.unican.es/.

[129] VaST Systems Technology Corporation. *CoMET – Electronic System-level Design Environment*. http://www.vastsystems.com/products_comet.html.

[130] Vector Informatic GmbH. *CANAlyzer – The Tool for Comprehensive Network Analysis*. http://www.canalyzer.com.

[131] *Virtual Socket Interface Alliance*. http://www.vsi.org.

[132] Volcano Communications Technologies AB. *Volcano Network Architect for CAN*. http://www.volcanoautomotive.com/products/can.htm.

[133] F. Wolf. *Behavioral Intervals in Embedded Software – Timing and Power Analysis of Embedded Real-Time Software Processes*. Kluver Academic Publishers, Boston, 2002.

[134] F. Wolf and R. Ernst. Intervals in software execution cost analysis. In *Proceedings of the IEEE/ACM International Symposium on System Synthesis*, pages 130–135, Madrid, Spain, September 2000.

[135] F. Wolf, J. Kruse, and R. Ernst. Segment-wise timing and power measurement in software emulation. In *Proceedings of the IEEE/ACM Design, Automation and Test in Europe Conference, Designers' Forum*, Munich, Germany, March 2001.

[136] J. Xu and D. L. Parnas. Priority scheduling versus pre-run-time scheduling. *Real-Time Systems*, 18(1):7–24, 2000.

[137] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *International Conference on Computer Design*, 1995.

[138] T. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(11), November 1998.

[139] W. Zhao and J. Stankovic. Performance analysis of fcfs and improved fcfs scheduling algorithms for dynamic real-time computer systems, 1989.

[140] D. Ziegenbein. *A Compositional Approach to Embedded System Design*. PhD thesis, Technical University of Braunschweig, 2003.

[141] D. Ziegenbein, R. Ernst, K. Richter, J. Teich, and L. Thiele. Combining multiple models of computation for scheduling and allocation. In *Proceedings Sixth International Workshop on Hardware/Software Co-Design (Codes/CASHE '98)*, pages 9–13, Seattle, USA, March 1998.

[142] D. Ziegenbein, M. Jersak, K. Richter, and R. Ernst. Breaking down complexity for reliable system-level timing validation. In *Ninth IEEE/DATC Electronic Design Processes Workshop (EDP'02)*, Monterey, USA, April 2002.

[143] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele. Representation of process mode correlation for scheduling. In *Proceedings International Conference on Computer-Aided Design (ICCAD '98)*, San Jose, USA, November 1998.

[144] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich. SPI – A system model for heterogeneously specified embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4), August 2002.

[145] D. Ziegenbein, J. Uerpmann, and R. Ernst. Dynamic Response Time Optimization for SDF Graphs. In *Proceedings International Conference on Computer-Aided Design (ICCAD '00)*, San Jose, November 2000.

[146] D. Ziegenbein, F. Wolf, K. Richter, M. Jersak, and R. Ernst. Interval-based analysis of software processes. In *Proceedings Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES '2001)*, Snowbird, USA, June 2001.

[147] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Swiss Federal Institute of Technology Zurich, 2001.


CAROLO-WILHELMINA
BRAUNSCHWEIG